TRƯờNG ĐạI HọC Kỹ THUậT THÀNH PHố Hồ CHÍ MINH
KHOA CÔNG NGHệ THÔNG TIN

LUÂN VĂN TÔT NGHIệP

<u>NộI DUNG Đề TÀI :</u>

TÌM HIểU NGÔN NGữ VHDL. VIếT CHUONG TRÌNH THIếT Kế MạCH CộNG 8 BIT SONG SONG BCD CHO 2 TOÁN HạNG

GIÁO VIÊN HƯớNG DẫN : THầY NGUYễN QUốC TUấN.
SINH VIÊN THựC HIệN : ĐặNG MINH TUấN

Năm 1999

Lời GIới THIệU

Máy tính có vị trí quan trọng trong cuộc sống,phép cộng (adding) là phép toán thường gặp trong các công việc hằng ngày cũng như trong các bài toán kỹ thuật, kinh tế nhằm để tính toán một cách nhanh chóng .Việc hiểu biết thông các kết cấu hệ thống máy tính là quan trọng,nghĩa là nắm vững kết cấu về mặt phần cứng, tổ chức thiết kế mạch (các mạch điện bên trong hệ thống máy). Điều này dẫn đến một yêu cầu: cần có một ngôn ngữ lập trình có hệ thống đáp ứng yêu cầu này thông qua máy tính. Hiện đã có rất nhiều phần mềm ứng dụng trên lĩnh vực này, tuy nhiên phần lớn không có tính mềm dẻo, linh hoạt, lập trình chủ động trong thiết kế.

Một ngôn ngữ đã được ứng dụng và đang được phát triển mạnh được giới thiệu ở đây: Ngôn ngữ VHDL cùng với phần mềm ứng dụng Leonardo và Max+plus II.

Em tiến hành nghiên cứu các chức năng của ngôn ngữ VHDL cách sử dụng phần mềm Leonardo và Max+plus II để viết chương trình thiết kế mạch cộng 2 số BCD song song

*Em xin trân trọng cám ơn Thầy **Nguyễn Quốc Tuấn** và các thầy cô trong Khoa Công Nghệ Thông Tin Trường Đại Học Kỹ Thuật đã rất tận tình hướng dẫn và giúp đỡ em trong thời gian làm luận văn .*

PHầN I

CÁCH Sử DụNG PHầN MềM

LEONARDO VÀ MAX+PLUS II

A. <u>CÁCH Sử DụNG LEONARDO.</u>

1. Lời mở đầu:

Exemplar logic's Leonardo công cụ tổng hợp tối ưu và phần tích mạch logic một cách linh hoạt và có tính tác động lẫn nhau , đã phát triển để cho phép sử dụng các thiết kế công nghệ độc lập : ASIC,FPGA và CPLD. Người thiết kế có thể củng cố những thiết kế đã làm sang một thiết kế khác , bảo quản và vận dụng thứ bậc thiết kế ,sử dụng ngôn ngữ mô tả phần cứng (VHDL) để thể hiện thiết kế của chúng .

Bộ sách hướng dẫn sử dụng leonardo bao gồm :

Leonardo User Guide giới thiệu cách dùng Leonardo và những dòng thiết kế có sẵn của nó.

Leonardo command Reference qui định chi tiết về cách sử dụng lệnh và biến.

Leonardo synthesis and Technology Guide qui định chi tiết về cách tổ hợp và tối ưu và cũng qui định cho chúng ta những thông tin để cài đặt công cụ trong công nghệ .

Leonardo User Guide có 8 thành phần :

Phần 1 : Giới thiệu

Phần 2 : Cách khởi động Leonardo (có 4 chế độ làm việc và 2 chế độ thực thi)

Phần 3 : Giới thiệu về Flow Guide giúp bạn khởi động thiết lập dòng thiết kế.

Phần 4 : Giao diện đồ hoạ (GUI) của Leonardo

Phần 5 : Vài điểm đặc biệt trong giao diện dòng lệnh (command line)

Phần 6 : Vắn tắt những lệnh trong Leonardo, các bước cần thiết kế để xây dựng dòng thiết kế .

Phần 7 : Mô tả cách mà Leonardo lưu trữ dữu liệu thiết kế.

Phần 8 : Mô tả cách xem qua thứ bậc của thiết kế .

Leonardo là một môi trường thiết kế có tính thứ bậc ảng hưởng lẫn nhau bao gồm các tối ưu ràng buộc . Leonardo bảo quản thứ bậc và cho phép chuyển đổi qua lại hoặc tạo ra các thứ cấp thứ bậc để tối ưu kết qủa , ta phải chi tiết hoá cho thông tin cho mỗi cấp thứ bậc . Leonardo có thể chạy được ở chế độ INTERACTIVE hoặc công cụ BUTTON. Khi chạy chế độ INTERACTIVE chúng ta thực hiện :

- Xem qua thứ bậc
- Điều khiển ảnh hưởng dòng thiết kế.
- Xem những thông báo về area và riêng ở bất kỳ cấp nào .
- Lập ràng buộc và tối ưu ở mọi cấp
- Xem mạch và đường dẫn .

2. Khởi động leonardo:

Hai chế hoạt động :

a.**Interactive :** 2 giao diện.

- Giao diện trực tiếp
- Giao diện dòng lệnh .

*Batch (Bó):

- Script mode.
- Galileo compatibility mode

Tất cả những lệnh những lệnh có thể chạy từ Gui,dòng lệnh ở cửa sổ chính hoặc từ hộp thoại (Dialogue).Hơn nữ nó có thể chạy qua lại từ Gui và dòng lệnh hoặc từ bằng bó (batch) dùng Script file.

* ***Chế độ Interactive:***

-Graphical user interface : Gõ vào leonardo

-Gõ Leonardo – help để xem dòng lệnh :

+Nếu trong nền Unix hoặc dos gõ :leonardo

+Nếu trong nền window gõ Star Leonardo

- Dòng lệnh (Command line Interface) : Gõ **Elsyn**

Từ màn hình giao diệb trong Unix và Dos Shell chúng ta gõ lệnh : **Elsyn**

-Gõ lệnh **Elsyn – Help** để xem option dòng lệnh (lệnh mà không chạy được ở nền Unix nhưng chạy được trong nền Windows , chúng ta gõ Star **elsyn**). Leonardo tạo ra một shell TCL qui định từ dấu nhắc , chúng ta có thể nghiên cứu trạng thái thiết kế trạng thái chuyển đổi như đã mong muốn và chạy lại chi tiết của thiết kế với cách dùng thiết lập khác để xem những kết quả khác có thể thu được.

b. BATCH:

Script mode (user-defined flow):

Gõ : Elsyn - File [script-filr]

Leonardo sẽ chạy Tcl Script – file và thoát . Kịch bản (Script) sủ dụng bất kỳ lệnh đã được định nghĩa đầy đủ trong Leonardo và không cố định như với Galileo Compatibility.Tất cả các lệnh Tcl và lệnh có thể tìm thấy trong đường dẩn .

Dưới đây là ví dụ về File Script:

Load – Library X14

Read   my_file.vhd

Optimize - targertX14-eff quick

Write my_ file.xnf

Galileo compatibility mode:

Chạy mode Galileo Compatibility ở Leonardo trong môi trường Unix hoặc Dos shell ta gõ lệnh :

Elsyn   Input_file   Output_file argurments

Leonardo sau đó thực thi Tcl Galileo.scr

$ EXAMPLAR/data/galileo.scr

Mô tả này làm theo thái độ dòng lệnh cuả galileo gc

Ví dụ : elsyn my_file.vhd my_file.xnf –target X14 –effort quick

Lệnh tối ưu thiết kế VHDL trong file .vhd cho công nghệ Xinlinx 4000, với tối ưu nhanh , và ghi kết quả ra mảng my_file.xnf.

3.Flow guide:

Là công cụ giúp chúng ta có thể biết dòng lệnh trong thiết kế trong Leonardo. Ta có thể dùng chế dộ default hoặc do trình cài đặt (customize), cả hai hướng này đòi hỏi ở mỗi bước phải tuần tự và quy định những thông tin để chạy mỗi lệnh . Mỗi bước mỗi hộp thoại (Dialog Box) chúng ta phải chỉ rõ lệnh chọn .Bản thân lệnh đã được thể hiện ở cửa sổ chính Leonardo , nơi mà output từ lệnh cũng được hiển thị .

Chúng ta có thể học hai cách dùng : Dialog Box và dòng lệnh .

* *Running the Flow guide :*

Để sử dụng flowGuide , Click tuần tự trên mỗi Button để hiển thị hộp thoại cũa mỗi lệnh. Khi hộp thoại đã được chọn,1 mô tả vắn tắt của hộp và cách sử dụng của nó được hiển thị trong Flowguide .

Để chạy mỗi lệnh ta đưa vào những thông tin yêu cầu và chọn button của hộp thoại . Lệnh và output của nó sẽ được hiện thị trên cữa sổ chính một cách chính xác như chúng ta gõ vào đó .
Khi hoàn thành mỗi lệnh thì nhắp button kế tiếp theo một trình tự .

* *Customize flowguide :*

Hộp thoại này cho phép chúng ta chọn bất kỳ item nào cho thiết kế .Để dể hiểu hơn về mỗi item ảnh hưởng ra sau ta dời currsor qua item đó để hiện một bảng help hình cầu với thông tin về Flowguide sẽ được bổ sung nếu item đó được chọn . Sau khi được chọn tất cả item mong muốn , nhấn button Run flowguide để hiện Flow Guide của mình . Nếu không có item nào thay đổi chúng ta sẽ thấy flowguide default.Để lưu Flow guide của mình chọn Save Setting Now từ Menu Option (hoặc chọn Save Setting On Exit). Lần sau Flow guide sẽ là Default. Chúng ta có thể bật tắt Customize Flow Guide từ Change Preferrence .

4.Cách dùng Gui:

* *Cửa sổ chính Leonardo:*

Cho chúng ta xữ lý dòng lệnh leonardo và lệnh Tcl khác ,cũng như vài lệnh của hệ thống như cd,pwd… và lệnh khác có thể tìm thấy trong đường dẫn .

+ Dùng phím mũi tên để dùng chạy những lệnh trước đó .
+ Gõ và lệnh :(sử dụng được Cut và Copy,Paste)
+^w:xoá từ trước đó ,^a về đầu dòng .
+Tất cả lệnh xây dựng c ách dùng hợp lệnh được hiện thị như là input,nếu chúng ta đã gõ chúng (và cũng cài sẳn trong tập lệnh mà chúng ta có thể cuốn qua bằng bàn phím mũi tên).

* ***File menu:***

Danh sách các option trong file menu như sau:
+Edit file : màn hình soạn thảo Leonardo
+ Run Script : nguồn kịch bản Tcl (lệnh Tcl Source filename-filename được chúng ta chọn) .
+Save stranscript:lưu Transcript (toàn bộ văn bản được hiện thị trong cữa sổ chính Leonardo bao gồm input và output) - Lệnh này được dùng ở bất kỳ thời điểm nào
và phải chỉ rõ tên file mà chúng ta muốn lưu . Có hạn chế tối đa số dòng được hiện thị
trong transcript (default=1000 dòng)
+Clear transcript : Xoá tất cả dòng lệnh trong transcript
+Exit : thoát khỏi leonardo

* *Command menu :*

Mỗi item trong :I/O,Optimize,report và Hierachy hiển thị ra hộp đối thoại cho phép chúng ta thiết lập các thông số của lệnh và chạy lệnh nó .Lệnh mà được xây dựng trong hộp thoại được thể hiện trong dòng lệnh kèm theo bất kỳ output từ lệnh đó .

+ I/O command menu:
Load library :nạp thư viện công nghệ (Actel,Flex,..)
Read :đọc các file nguồn (VHDL,verilog,…)
Load modgen :nạp khối phát sinh Modgen
Write :ghi file đích
+*Optimize command menu:*
+*Report command menu:*
+*Hierachy command menu:*
+*Tool menu:*

Tất cả các option trong tool menu như sau:

Flowguide : hiển thị Flowguide hoặc customize flowguide
Design Browser: Hiển thị *Design Browser*: Hiện thị mạch
Schematic viewer: hiện thị mạch.
Constraint editor: hiện trình soạn thảo ràng buộc cho phép chúng ta có thể lập các thuộc tính (ràng buộc) trên bất kỳ đối tượng nào trong thiết kế .
Convenience Procedures: hiện thị hộp thoại Convenience Procedures cho phép truy xuất 1 vài thủ tục Tcl-nơi đã qui định để tạo ra những công việc dễ dàng thực hiện .
+Option menu :

Các mô tả của ***Option Menu*** như sau:

Change Preferrences: Cho phép chúng ta cài cách dùng GUI
Baloon Help :Nếu được chọn bảng help hình tròn ,được hiển thị bát cứ lúc nào tại vị trí curror.
Toolbar :hiện toolbar tổng quát nhất .Có thể thay đổi vị trí cũ toolbar trong ***"option Change references"***
Chú ý phải lưu cách thiết lập đó nếu chúng ta muốn cách định vị này cho lần khởi động leonardo sau .
Save setting on exit: tất cả các thiết lập khi thoát khỏi Leonardo được lưu lại trong lần sau.
Save setting now: lưu các thiết lập hiện thời với filr cấu hình leonardo.ini
+Help menu:
Cho phép chúng ta hiện thị các thông báo help cho tất cả :
- Lệnh :(tương đương gõ help command)
- Biến :(tương đương gõ help-variables)

**** Customizing the gui :***
+ Change Preferences
Chức năng của các option trong hộp thoại như sau:
- ***Show about box at starup*** : hiện thị about box mỗilần khởi động gui
- ***Ask to save transcript***

Before delete : nhắc nhở bạn lưu các transcript khi số dòng vượt quá số dòng qui định tối đa trước khi tự động xoá
Show customize flow
Guide screeen before flow guide:hiện hộp đối thoại flowguide khi bạn chọn button flow guide cữa sổ chính .
Toolbar Position:vi trí ngầm định cho các toolbar
-Theo chiều dọc nếu chọn left hoặc right.
- Theo chiều ngang nếu chọn top hoặc button.

Chú ý :Nếu toolbar đã được hiện thị khi item đã bị thay đổi
.Max lines :
Chỉ rõ dòng quy định tối đa được hiển thị ở cửa sổ chính trước khi xoá transcript .
Default =1000. Khi vượt quá giới hạn này, bạn được nhắc để lưu lại transcript trước khi bị xoá 10% cuối cùng của dòng vượt quá, bạn vẫn có thể thấy được ở output gần nhất .
.Editor :sử dụng khi chúng ta chọn "edit file" từ file menu trong màn hình cữasổ chính .
Window color :Cho biết màu nền cửa sổ, có thể đưa vào bất kỳ giá trị trong mỗi văn bản hoặc ở dạng thập lục phân . Cũng có thể cho giá trị từ Button Listbox, nếu click đúng bất kỳ Item nào trong lúc "***Set Color Type*** ", màn nền của ***Listbox*** sẽ thay đổi , vì thế chúng ta có thể thấy được màu nào thích hợp.
.Input Textcolor : Cho biết màu của văn bản trên cửa sổ
.Prompt color : Cho biết màu của thông báo lỗi .Lập "set color type"=error

.Set color type : cho biết giá trị màu để thay đổi khi chúng ta chọn một sự lựa chọn trong listbox và không làm thay đổi màu nền hoặc màu văn bản.

****Bổ sung file khởi động :***

Khi chọn **Option** *save setting now* (hoặc khi chúng ta thoát khỏi Gui với Option Save Setting On Exit tất cả các hiện thời được lưu vào File Leonardo.ini trong thư mục khởi động .Chúng ta có thể cài đặt file cục bộ này bằng cách thêm vào bất kỳ lệnh Tcl nào chúng ta cần .Một ví dụ chung sẽ được thêm vào hộp thoại thư viện Load Library. File toàn cục $EXAMPLAR/data/leonardo.ini luôn luôn ở gốc khi khởi động . Nếu ngẫu nhiên xảy ra một tk/tcl mã nguồn đã được định vị trong thư mục $EXAMPLAR/data/leonardo.ini và cũng đã được bổ sung .

** Thêm vào một thư viện :*

Dùng hộp thoại load library.Hãy soạn thảo file leonardo.ini trong thư mục khởi động (chọn option /save setting now nếu nó không hiện diện trước đó) và thêm vào 1 dòng lệnh dưới đây vào phần trên:

Append glbvar_priv(techlist)

“mem nam “ library_name_modgen_library_nam “lib cype:11 11”.

5.Giao diện dòng lệnh :

Phần này mô tả về giao diện dòng lệnh   , nó cho phép :

*Thực hiện các bước độc lập của quá trình tổng hợp .

*Đọc các thiết kế   vào trong cơ sở dử liệu leonardo .

*Viết đủ các thiết kế .

*Nạp các công nghệ và các công nghệ tạo ra mo dun.

*Tối ưu hoá các thiết kế trong cơ sở dử liệu cho công nghệ đặt biệt.

*Tạo area và các báo cáo định thời.

Các lệnh được nhập vào ở dấu nhắc khi ở chế độ tương tác hoặc có thể được lưu trong file và sau cùng được dùng như 1 kịch bản .

Chú thích : Tất cả các lệnh có thể được nhập vào từ cữa sổ chính leonardo GUI cũng giống như dòng lệnh elsyn(non GUI) và kịch bản Tcl.

Lệnh help:

Có thể hiện thị thông tin về các dòng lệnh bằng cách dùng lệnh help .Lệnh help dùng biểu thức hợp lệ (tên có có hoặc không có ký tự liên kết) và in cách sử dụng cho các lệnh mà sắp xếp biểu thức hợp lệ.

Ví dụ :help pre* sẽ hiện thị thông tin về tất cả các lệnh mà bắt đầu với chuỗi pre

Bổ sung lựa chọn và lệnh tự động :

Giao diện dòng lệng của leonardo có sự bổ sung lệnh tự động :Ta không ohải gõ đầy đũ lệnh khi mà nghĩa có nó không bị cấm .

Các lệnh Leonardo thường có nhiều tuỳ chọn , lựa chọn thực hiện sự bổ sung tự động bạn không cần nhập vào đầy đủ .

Nặc danh (Aliasing)

Leonardo đưa ra lệnh Alias cho phép các chúng ta có thể xác định tên riêng cho các chuỗi lệnh được dùng chung .

Alias Lp List Design - Port

Biến

Leonardo hổ trợ ngôn ngữ Tcl. Do đó phép gán biến và sự xác định phù hợp vói cú pháp Tcl (set cho thiết lập biến và $ var_name để định vị trí 1 biến)

Có 1 tập hợp những biến có liên kết trực tiếp đến máy tổng hợp leonardo. Những biến này có ảnh hưởng đến cách vận hành của lệnh tổng hợp. Danh sách biến leonardo hiện thị đầy đủ khi dùng lệnh Help_Variables (Help –V là đầy đủ)

Cài đặt giao diện dòng lệnh :

Leonardo nạp kịch bản dưới đây khi khởi động :

$EXEMPLAR/data/leonardo.ini

Có thể cài đặt giao diện dòng lệnh bằng cách điều chỉnh file exemplar.ini. Thông thường dùng những alias và các thủ tục Tcl như View_Schematic và Push_Design đã được xác định trong file này có thể thêm vào định nghĩa cuả mình trong file này. Nếu đã có file Examplar.ini trong thư mục cục bộ Leonardo nạp file thay vì trong lúc khởi động là $EXAMPLAR/data/leonardo.ini.

6.Các dòng dữ kiệu :

Phần này mô tả một số lệnh cho phép xây dựng các dòng thiết kế .Nếu sử dụng GUI có thể xem ***Flowguide*** một công cụ học từng bước xuyên suốt dòng tổng hợp . Để có thể chi tiết được cách sử dụng một lệnh đánh vào ***Help command name*** hoặc tham khảo sổ tay

Leonardo command reference.

Các kỹ thuật Loading :

Bắt đầu bất kỳ dòng thiết kế nào , nạp các thư viện công nghệ mà ta cần thiết nạp cho công nghệ đích để tổng hợp ,và công nghệ nguồn với một dảy (netlist) định rõ công nghệ .Ta dùng lệnh load_library để nạp công nghệ , ví dụ :

Load_library act3

Lệnh này nạp vào công nghệ Actel Act3 vào trong cơ sở dữ liệu leonardo.

Một danh sách đầy đủ các công nghệ được cung cấp bởi Exemplar được liệt kê trong hộp thoại nạp thư viện cuả leonardo GUI. Các tập tin công nghệ được định vị trong $exemplar/lib.Có thẻ thu được cac thư viện công nghệ bổ sung từ những nhà bán hàng cung cấp hoặc các phòng thí nghiệm hopặc nhóm sưu tầm thứ 3.

Chuyển đổi exemplar nếu cần 1 thư viện công nghệ cần thiết mà không thoát khỏi cd.

Việc thể hiện thiết kế :

Kế tiếp đọc các tập tin thiết kế , nếu thiết kế là một tập tin đơn giản dùng lệnh read để đọc nó .Lệnh read có thể đọc những tập tin được định dạng VHDL, verilog ,edif và xnf .Nếu dùng các tập tin VHDL hoặc verilog và có thiết kế được lưu trong nhiều tập tin hoặc dùng các tập tin VHDL mà chỉ bao gồm các gói VHDL cần thiết đối với thiết kế thì tốt hơn nên dùng lệnh analyze đọc tập tin VHDL và verilog và jưu trữ nó trong cấu trúc dữ liệu trung gian lệnh analyze không tạo thiết kế ở thời điểm này . Chúng ta có thể phân tích các gói VHDL vẫn không chứa đựng một thiết kế hoặc thông số thực thể VHDL hoặc khối verilog .

Các lệnh phổ dụng :

Chúng ta có thể thực thi các lệnh sau khi đọc thiết kế , trước hoặc sau sự tối ưu :

* ***Write*** : ghi thiết kế đến các tập tin ,nếu ghi ra VHDL hoặc verilog trước khi tối ưu đạc tả công nghệ RTL style VHDL hoặc Verilog sẽ đươc ghi bao gồnm các phát biểu dòng dữ liệu ở mức thấp.

* Report _area -all : lệnh này cho phép chúng ta tìm thấy nhiều thông tin về kích thước và sự phức tạp thiết kế trước sự tối ưu . Nó đếm tất cả các Cell của nút lá đặc tả công nghệ cung cấp số lượng các cổng AND, OR, DFF, MUX và các toán tử

**** Group/ungroup/unfold :***

Lệnh này thao tác phân cấp thiết kế.

**** View_schematic***

Lệnh này là đẻ xem sơ đồ Netscope và thể hiện thiết kế mức cao .Nó nạp tự động thư viện ký hiệu các cell trong thiết kế .

Nếu chúng ta muốn đưa về mức gốc , dùng lệnh sau:

(ungroup –all –hierachy}

Để lưu một thiết kế đến 1 tập tin ghi nó ra 1 file edif.Có thể khôi phục thiết kế sau đó bằng cách đọc file edif.

Nếu muốn khôi phục một thiết kế trong một phần mới ,bảo đảm thư viện công nghệ đã được nạp đối với thiết kế được ánh xạ trước khi khôi phục thiết kế từ file edif .

Leonardo để lưu trử những thông tin không ẩn .Thông tin trong file edif đầy đủ để cài đặt lại thiết kế .

Sự tối ưu độc lập công nghệ nói chung chúng ta có thể thực thi một vài pre-optimization trên thiết kế trước khi nhắm vào bất kỳ công nghệ nào dùng lệnh pre-optimization. Lệnh pre – optimize-common-logic –unsigned.logic – extract thực thi sự lan truyền không đổi ,loại bỏ các biểu thức phụ chung ,loại bỏ các mạch logic không dùng và lấy counter/recorder ram.Lệnh này thực thi sự tối ưu độc lập công nghệ trên tất cả các mức phân cấp trong thiết kế . Nó không thay đổi sự phân cấp .Nó cũng không lan truyền xuyên suất sự phân cấp được định vị hoặc di chuyễn mạch logic các mức phân cấp .

Sự tạo khối :

Chúng ta có thể thực hiện tất cả các tác vụ cuả các toán tử , với các cổng ở mức thấp bằng bởi cách dùng lệnh Resolve_modgen.Nếu muốn dùng các bộ tạo khối đặc tả công nghệ để thực hiện trong các toán tử , thực thi lệnh load một kênh đầu tiên.

Sự tối ưu đặc tả công nghệ :

Dùng lệnh optimize sẽ ánh xạcác mạch logic đến các cell công nghệ đích hoặc các bảng tra cứu và bổ sung bộ đệm I/O đến thiết kế mức đỉnh lệnh này không thay đổi ranh giới phân cấp nhưng nó sẽ thực thi tối ưu trên mức của mỗi

Phân cấp với các tuỳ chọn effort=quick cho kết quả nhanh nhất và các tối ưu các khu vực nhỏ .

Ngoài ra còn các điều kiện:

Xác lập các ràng buộc :

Sự tối ưu thời gian:

Sự chuẩn bị cho soạn thảo:

Các công nghệ LUT:Xilinx

7. The design database:

Gồm các mục sau:

7.1-Thiết kế mô hình thông tin data:

7.2-Truy xuất data thiết kế :

8.Sử dụng xem nhanh thiết kế :

Library windows:

Hierachy windows:

Thể hiện dòng đối tượng được lựa chọn và thiết kế hiện tại :

Các nút cuối màn hình(bottom row of button):

8.1-Việc lựa chọn các đối tượng :

8.2-Xác lập các đối tượng hiện tại :

8.3-Sự thao tác các đối tượng thể hiện:

8.4-Bổ sung 1 view đến cửa sổ hierachy:

8.5-Tháo bỏ 1 view cữa sổ hierachy:

8.6-Cập nhật design browser :

8.7-Thông tin trong cột cửa sổ hierachy:

Cữa sổ hierachy thể hiện 2 cột :

- Cột 1: thể hiện sự mô tả graphic/text của các đối tượng thiết kế .
- Cột 2: các thông tin bổ sung về 2 loại đối tượng : các instance và port.

B.CÁCH SỬ DỤNG MAX+PLUS II

*** TổNG QUÁT:**

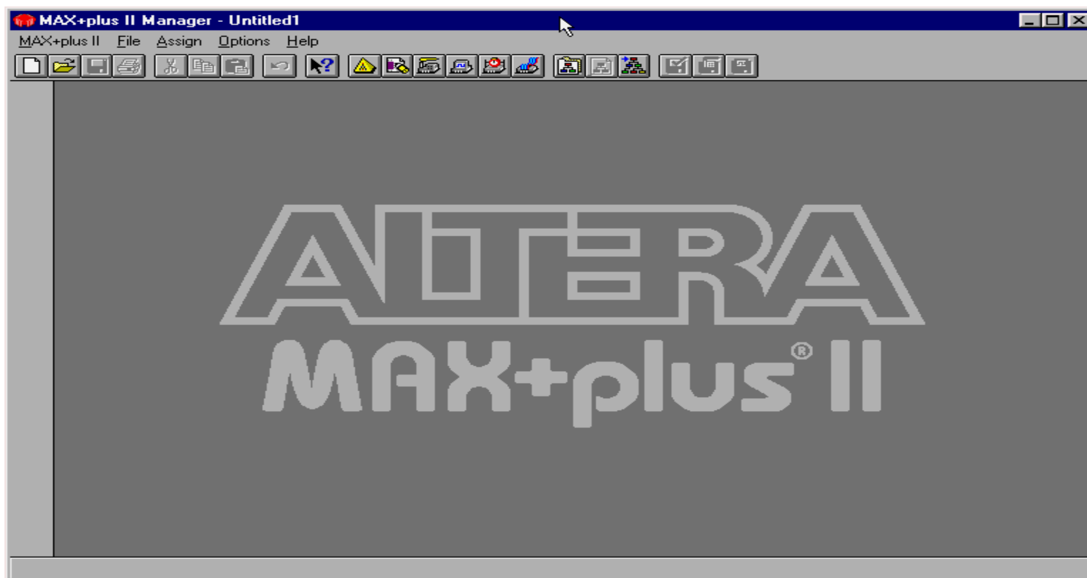
Trong qúa trình thiết kế và tổng hợp mạch một yêu cầu đặt ra cuối cùng là kiểm tra được mạch đã thiết kế có kết qủa đúng theo yêu cầu . Trong các phầm mềm về tổng hợp và kiểm tra mạch có phần mềm Max+plusII là một trong những phần mềm rất mạnh về tổng hợp và kiểm tra mạch . Sau đây tôi trình bày cách sử dụng về phần mềm này :

1. Khởi động phần mềm Max+plusII :

Sau khi phần mềm Max+plus II đã được cài đặt ta tiến hành khởi động như sau :

- **Bước 1** : Bấm vào biểu tượng *MAX2WIN*

Sau khi bấm vào biểu tượng để khởi động màn hình Max+PlusII hiện ra như sau:



Trình bày của màn hình Max+ Plus2 gồm các thành phần sau:

-*Dòng 1* :Thanh Toolbar dùng hiện thị thư mục và tập tin hiện hành .

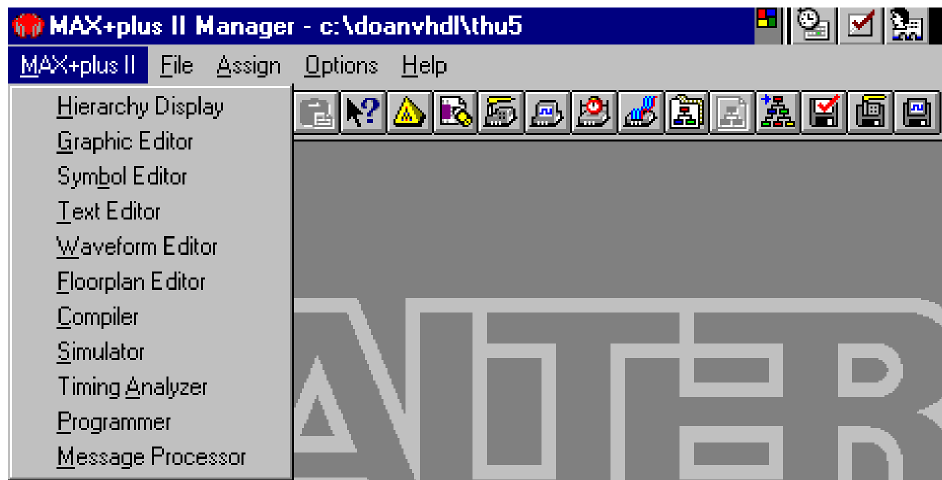
-*Dòng 2* : Gồm 5 mục chính :

+ Maxplus2
+ File
+ Assign
+ Option
+ Help

- *Dòng 3* : các biểu tượng sử dụng trực tiếp các ứng dụng của Max+plus2
- *Dòng 4* :Khu vực trình bày các loại text.

- **Bước 2** : Các thành chính trong các mục dòng 2 như sau :

a-*Mục Max+plusII :*



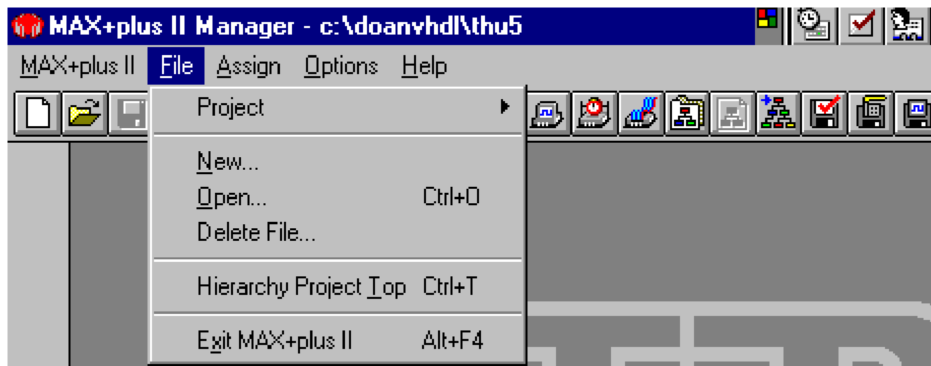
Maxplus2 đưa ra nhiều giúp đỡ. Hệ thống bao gồm 11 nhóm ứng dụng tổng hợp dùng để thiết kế đến thực thi chương trình.

Có 11 phần :

1. Hierarchy Display : trình bày các phân cấp sau khi thiết kế và dịch.
2. Graphic editor :cho chế độ vẽ mạch thiết kế và test mạch theo dạng cấu trúc
3. Symbol Editor: đưa ra một thực thể entity sau khi đã dịch chương trình.
4. Text Editor :vào soạn thảo chương trình
5. Waveform Editor:soạn thảo và test mạch của chương trình
6. Floorplan Editor:soạn thảo và thiết kế theo dạng Floorplan
7. Compiler: dịch chương trình
8. Simulation:thực hiện mô phỏng
9. Timing Analyzer: phân tích thời gian của chip
10. Programmer: xác định của thiết bị phần cứng
11. Message processor:màn hình thông báo và ánh xạ đến các lỗi khi dịch .

B - Mục file :

Mục file gồm có 6 thành phần :



1-Project:xác định chương trình cần dịch và ghi …

2-New: mở một file mới của các chế độ :text,graphic,symbol,waveform editor

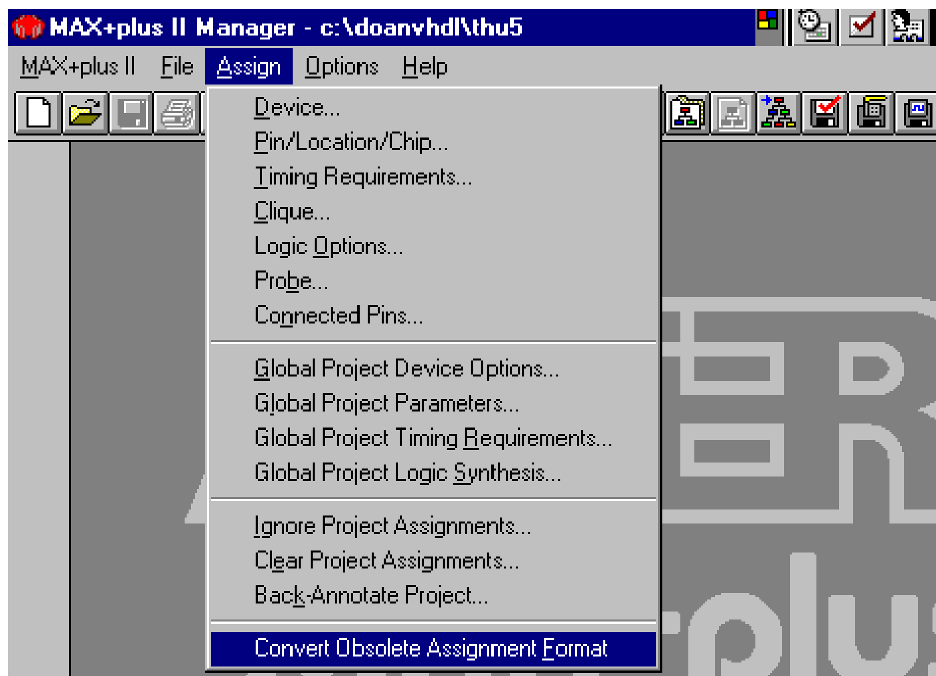
3-Open: mở các file text,graphic,symbol,waveform editor

4-Delete file:xóa các file

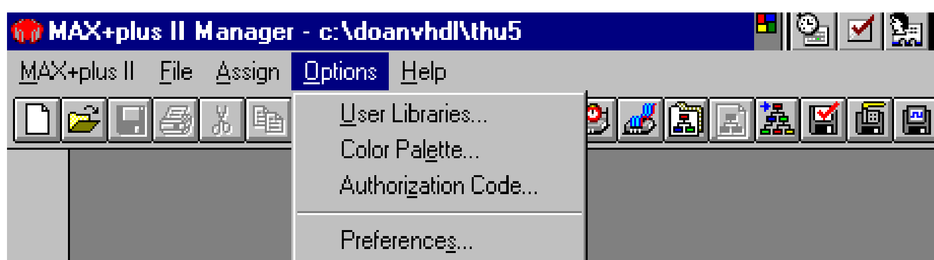
5-Hierachy Project Top:chuyển về chương trình nguồn đã đựoc lựa chọn dịch

6-Exit Max+plusII:thoát khỏi max+plus2

c-Mục Assign:gồm các thành phần :



d-Mục Option:gồm các thành phần :



2. Thực hiện soạn thảo và dịch một chương trình :

Để thực hiện soạn thảo và dịch 1 chương trình ta thực hiện theo trình tự sau:

- *Để soạn thảo một chương trình tổng hợp phần cứng :*

  Vào File \New\Text editor file

Khi này màn hình soạn thảo xuất hiện ,việc soạn thảo trong môi trường này giống như trình soạn thảo Winword. Sau khi soạn thảo xong ghi tập tin và đặt tên tập tin có phần mở rộng là **.vhd .**

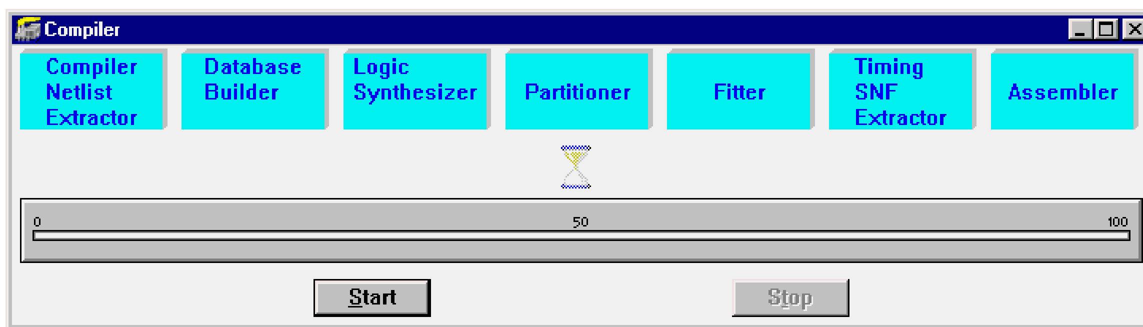
- *Để dịch một chương trình đã được soạn thảo ta theo các bước sau:*

Xác định tập tin hiện tại cần dịch bằng cách thưc hiện :

Chọn File\Project\ set project to current file .

Sau khi đã chọn xong theo yêu cầu trên ta tiến hành dịch:

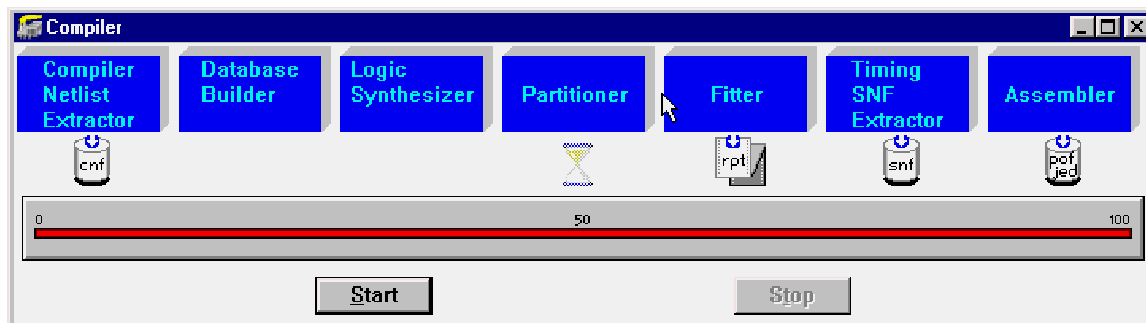
-Chọn mục **MAX+plus \ COMPILER** khi này cửa sổ của COMPILER sẽ xuất hiện như hình sau:



Màn hình Compiler hiển thị nhiều mục khác nhau ,đó là những bộ phận của những quá trình dịch và tổng hợp mạch .

Chọn nút **Start** để bắt đầu quá trình dịch. Trong qúa trình dịch các mục duyệt qua, nếu thành công thì các mục sẽ được duyệt hết và hiện thị một thông báo về các thông tin trong quá trình dịch .

Màn hình sau sẽ thông báo khi quá trình dịch thành công :



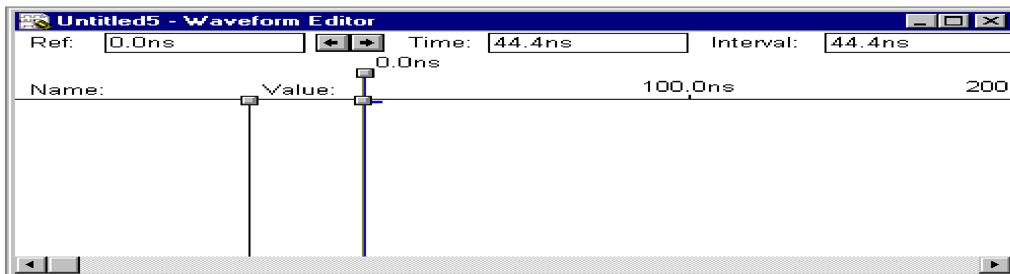
3. Thực hiện kiểm tra kết qủa sau khi đã tổng hợp mạch :

Sau khi dịch xong chương trình tổng hợp mạch ta tiến hành kiểm tra kết quả của mạch được tổng hợp .Việc kiểm tra được tiến hành như sau:

a .Soạn thảo một tập tin file waveform editor file :

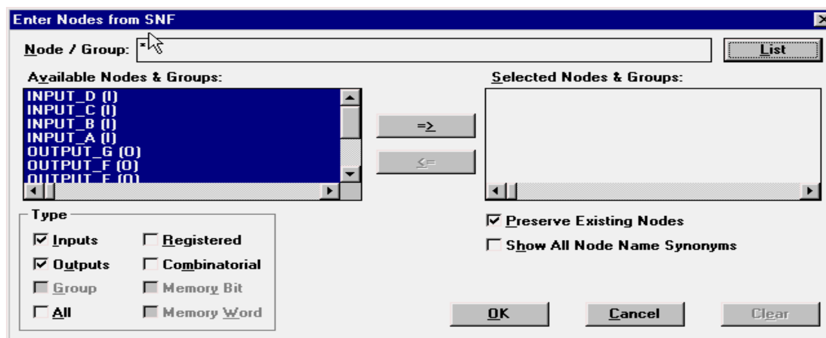
Chọn File\New \Waveform editor file.

Khi này màn hình soạn **Waveform editor file** xuất hiện như sau:



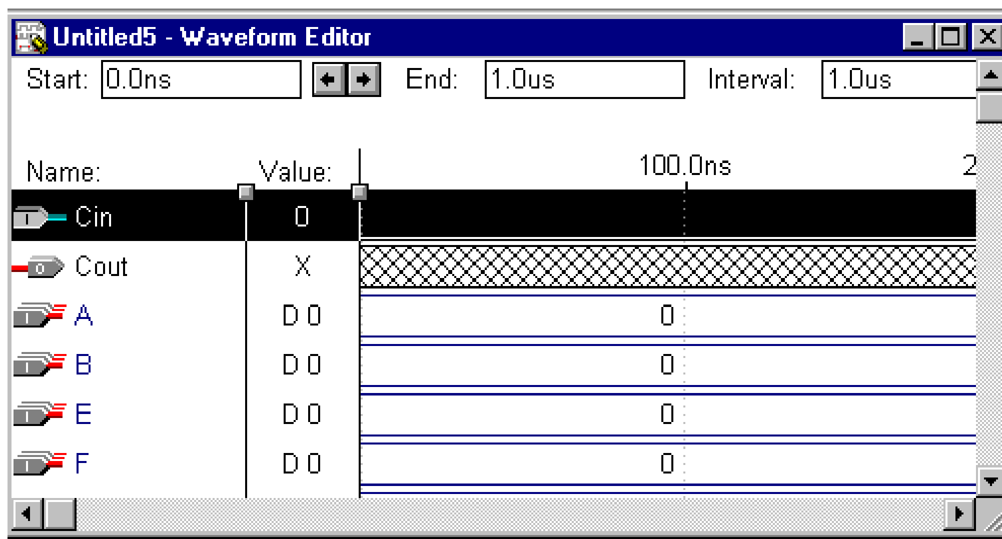
Để đưa các thành phần cần kiểm tra của mạch ta thực hiện chọn :

Chọn mục Node \enter Nodes from SNF khi đó màn hình của các thành phần mạch sẽ xuất hiện như sau :



Ta tiến hành chọn nút **List** và đưa các biến của Node sang màn hình Waveform sau đó chọn nút **OK.**

Màn hình Waveform sau khi được chọn các biến có dạng như sau :

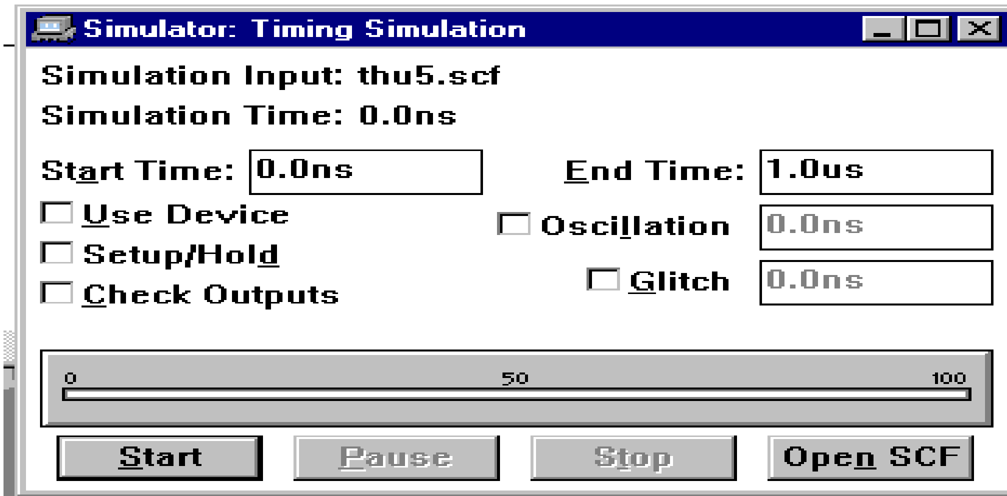


Sau khi đã chọn xong các biến trong màn hình Waveform Editor ta lưu (ghi) tên tập tin này với phần mở rộng . scf .

b .Tiến hành mô phỏng kết qủa trên màn hình Waveform Editor.

Chúng ta có được tập tin waveform đã được nhập ở trên, chúng ta có thể mô phỏng bảng thiết kế và xác định nó làm việc đúng hay sai.

Mở bảng mô phỏng bằng cách chọn Max+plus II \ Simulator bạn sẽ nhìn thấy cửa sổ bảng mô phỏng được mở như hình dưới đây:



Chọn **Start** để bắt đầu thực hiện mô phỏng. Bảng mô phỏng sẽ hoàn thành. Và đưa ra kết qủa trong bảng waveform editor. Kết quả đúng hay không phụ thuộc vào chương trình của người thiết kế tổng hợp mạch.

Tất cả các bước trên có thể được thực hiện được nhiều lần cho đến khi nào đám ứng đúng yêu cầu của người thiết kế và tổng hợp mạch .

PHầN II

NGÔN NGữ VHDL

Chương 1 : PHẦN GIỚI THIỆU

1.1 VHDL LÀ GÌ ?:

VHDL là chữ đầu của ngôn ngữ mô tả phần cứng VHSIC (VHSIC là chử đầu của Very High Speed Integrated Circuits) . Đó là ngôn ngữ mô tả phần cứng có thể sử dụng mẫu hệ thống digital ở nhiều mức ý tưởng từ thuật toán đến cổng logic . Sự phức tạp của hệ thống digital là kiểu có thể biến đổi từ cổng đơn giản đến hệ thống điện tử phức tạp. Hệ thống digital có thể mô tả 1 cách thứ tự , cần phải có kiểu rõ ràng trong các mô tả giống nhau .

Do đó ngôn ngữ có cấu trúc cho phép biểu diễn hành vi đồng thời hoặc tuần tự của hệ thống digital với sự điều chỉnh bên ngoài . Nó xem hệ thống như là kết nối chung quanh các thành phần .Kiểm tra các dạng sóng có thể sử dụng các cấu trúc giống nhau .

Ngôn ngữ không chỉ định nghĩa cú pháp mà còn định nghĩa mô phỏng cho mỗi cấu trúc ngôn ngữ . Do đó các kiểu đã viết trong ngôn ngữ có thể sử dụng thẩm tra việc mô phỏng VHDL . Đó là kiểu viết rõ ràng và dài dòng . Nó thừa kế nét đặc trưng ngôn ngữ tuần tự , từ ngôn ngữ lập trình Ada . Mô tả đầy đủ phải có khả năng mô tả nhiều chip phức tạp tới hệ thống điện tử đầy đủ .

1.2 CÁC CHỨC NĂNG :

Sau đây là các chức năng chủ yếu mà ngôn ngữ cung cấp đi đôi với những điểm đặc trưng từ các ngôn ngữ mô tả phần cứng khác nhau .

+ Ngôn ngữ có thể xem như là nơi trao đổi trung gian giữa các Vendor chip và người sử dụng công cụ CAD . Các Vendor chip khác nhau có thể mô tả VHDL với các thành phần thiết kế hệ thống . Người sử dụng cung cấp CAD có thể sử dụng nó để nắm bắt các hành vi của thiết kế tại mức cao của sự trừu tượng quá trình mô phỏng .

+ Ngôn ngữ có thể sử dụng truyền tin trung gian giữa các công cụ CAD và các CAE khác nhau . Ví dụ , lược đồ của chương trình có thể sử dụng bộ mô tả thiết kế VHDL , có thể sử dụng chúng như là input cho quá trình mô phỏng .

+ Ngôn ngữ hổ trợ cho Hierarchy , 1 hệ thống digital có thể hiểu như là tập hợp các thành phần kết nối chung quanh , mỗi 1 thành phần vào có thể hiểu như tập hợp của các thành phần con kết nối chung quanh .

+ Ngôn ngữ hổ trợ phương pháp thiết kế linh động từ trên xuống , từ dưới lên hoặc hỗn hợp .

+ Ngôn ngữ không là kỷ thuật đặc trưng , nhưng nó có khả năng hổ trợ các kỷ thuật đặc biệt đó . Nó có thể hổ trợ cho các kỷ thuật phần cứng khác nhau , ví dụ có thể định nghĩa các kiểu logic mới và các thành phần mới , bạn có thể xác định rỏ các tính chất của kỷ thuật đặc trưng ,bởi vì nó là kỷ thuật độc lập. Các kiểu giống nhau có thể tổng hợp đưa vào các thư viện khác nhau.

+ Nó hổ trợ cho cả hai kiểu thời gian đồng bộ và không đồng bộ .

+ Các kỷ thuật digital khác nhau ,như là các mô tả trạng thái kết thúc , các mô tả tính toán , và các phương trình boolean , có thể là kiểu sử dụng ngôn ngữ .

+ Ngôn ngữ có thể dủng 1 cách công khai , có thể đọc được bởi người , bởi máy móc và trên tất cả là nó không có người sở hữu .

+ Nó là chuẩn IEEE và ANSI , do đó các kiểu mô tả sử dụng cho ngôn ngữ này được linh động . Cần quan tâm nhiều hơn trong việc bảo trì chuẩn đã thu được và từ từ có thể hình thành chuẩn thứ hai .

+ Ngôn ngữ hổ trợ cho 3 loại mô tả cơ bản khác nhau : cấu trúc , dòng dữ liệu và hành vi . Một thiết kế có thể được biểu diễn trong mỗi sự kết hợp của các loại mô tả trên.

+ Nó hổ trợ các mức ý tưởng rộng , từ ý tưởng mô tả hành vi tới mô tả chính xác các mức cổng .

+ Tùy các thiết kế có thể có kiểu sử dụng ngôn ngữ và chúng không bị hạn chế bởi kích thước của thiết kế .

+ Ngôn ngữ có nhiều phần tử tạo thành kiểu thiết kế theo phạm vi , ví dụ các component , các function , các procedure và các package .

+ Kiểm tra các cách viết sử dụng ngôn ngữ giống nhau tới việc kiểm tra các kiểu VHDL khác .

+ Generic và attribute được dùng trong mô tả các thiết kế theo tham số .

+ Một kiểu không cần mô tả hàm thiết kế nhưng có thể chứa các thông tin xung quanh bản thiết kế trong phạm vi sử dụng định nghĩa attribute , cũng như tổng hợp diện tích và vận tốc .

+ Một ngôn ngữ có thể sử dụng mô tả thư viện các thành phần từ các vendor khác nhau . Từ các công cụ đó sẽ không khó khăn trong việc hiểu các kiểu VHDL trong việc đọc các kiểu từ những trạng thái khác nhau của vendor từ ngôn ngữ chuẩn .

+ Các kiểu viết trong ngôn ngữ này có thể kiểm tra bằng việc mô phỏng từ các ngữ nghĩa mô phỏng đã định nghĩa chính xác cho mỗi cấu trúc của ngôn ngữ .

+ Các kiểu hành vi thích ứng với loại mô tả tổng hợp , là khả năng tổng hợp các mô tả cổng logic .

+ Khả năng định nghĩa các kiểu dữ liệu mới cung cấp khả năng mô tả và mô phỏng kỷ thuật thiết kế mới ở mức cao của ý tưởng trong việc thực thi các chi tiết.

1.3-Ý TƯỞNG PHẦN CỨNG :

VHDL sử dụng mô tả 1 kiểu của thiết bị phần cứng digital . Kiểu này chỉ rỏ cách nhìn bên ngoài của thiết bị và 1 hoặc nhiều cách nhìn bên trong . Cách nhìn bên trong của thiết bị chỉ rỏ theo hàm hay cấu trúc , trong khi cách nhìn bên ngoài chỉ rỏ các giao tiếp của thiết bị qua sự kết nối với các kiểu khác trong môi trường xung quanh . Hình 1.1 cho xem thiết bị phần cứng và kiểu phần mềm tương ứng .

Từ thiết bị tới thiết bị ánh xạ toàn phần 1-nhiều . Đúng vậy , 1 thiết bị phần cứng có thể có nhiều kiểu , ví dụ 1 kiểu thiết bị ở mức cao của ý tưởng có thể không các xung clock ở đầu vào , từ

đó xung clock không sử dụng trong việc mô tả . Ngoài ra data truyền ở giao tiếp bề mặt có thể xem như bị giới hạn , giá trị integer thay thế cho trị logic . Trong VHDL , mỗi kiểu thiết bị được xem như sự mô tả riêng biệt của 1 thiết bị duy nhất , gọi là entity . Hình 1.2 xem quan cảnh VHDL của 1 thiết bị phần cứng có các kiểu đa thiết bị , với mỗi một kiểu thiết bị mô tả 1 thực thể . Ngay cả thực thể 1 qua N mô tả N thực thể từ VHDL , trong thực tế chúng được mô tả thiết bị phần cứng như nhau . Thực thể là ý tưởng phần cứng của thiết bị phần cứng thực tế . Mỗi một thực thể được mô tả sử dụng 1 kiểu , bao gồm 1 quang cảnh bên ngoài và 1 hoặc nhiều cách nhìn bên trong tại 1 thời điểm , 1 thiết bị phần cứng có thể mô tả bằng 1 hoặc nhiều thực thể .

Chương 2   : DIỄN GIẢI THUYẾT TRÌNH

Chương này giới thiệu ngôn ngữ ,đặc điểm ngôn ngữ chủ yếu được mô tả trong chương này ,ở đây bạn có thể viết những mẫu VHDL đơn giản

2.1 THUẬT NGỮ CƠ BẢN:

VHDL là ngôn ngữ mô tả phần cứng ,có thể sử dụng cho mẫu hệ thống tín hiệu số .Hệ thống tín hiệu số có thể đơn giản là cổng logic hoặc phức tạp là hệ thống điện tử tổng hợp . Một ý tưởng của hệ thống tín hiệu này gọi là một thực thể trong chủ đề này . Một thực thể X , khi sử dụng trong thực thể khác làY thì nó là một thành phần cho thực thể Y .

Do đó ,một thành phần ngoài thực thể còn lệ thuộc vào thực thể nào.

Đối với mô tả một thực thể , VHDL cung cấp 5 kiểu khác nhau của thiết kế gốc , gọi là design units .Bao gồm:

1. Entity declaration.
2. Architecture body.
3. Configuration declaration.
4. Package declaration.
5. Package body.

Một thực thể là dạng sử dụng sự khai báo thực thể và tối thiểu phải có một phần thân architecture . Sự khai báo thực thể mô tảcái nhìn bên ngoài của thực thể , ví dụ gõ vào và gõ ra tín hiệu . Thân của architecture bao hàm việc mô tả bên trong của một thực thể ; ví dụ một tập hợp của các thành phần kết nối là biểu hiện cấu trúc của thực thể đó , hoặc là tập hợp các phát biểu đồng thời hoặc tuần tự biểu hiện hành vi của thực thể đó . Mỗi loại của sự biểu hiện được xác định trong các thân architecture khác nhau hoặc pha trộn với một kiến trúc đơn . Hình 2.1chỉ ra thực thể là một mẫu có thể tồn tại

Khai báo định dạng là sử dụng cho việc tạo định dạng cho thực thể . Được xác định bắt buộc của thân một kiến trúc từ nhiều kiến trúc có thể liên kết với thực thể đó , đó là phần bắt buộc để xác định thành phần sử dụng trong việc chọn một kiến trúc cho các thực thể khác . Một thực thể có thể có trong số những định dạng chung.

Khai báo gói tập hợp các khai báo liên hệ nhau , như là khai báo kiểu , khai báo kiểu con và khai báo chương trình con , phân phối qua hai hoặc nhiều thiết kế .Thân gói bao hàm định nghĩa của khai báo chương trình con bên trong khai báo gói.

Thực thể được khai báo một lần có hiệu lực qua hệ thống VHDL . Điển hình của hệ thống VHDL gồm sự phân tích và mô phỏng .

Phân tích là đọc một hay nhiều bản thiết kế bao gồm file gốc và file đã biên dịch từ thư viện thiết kế chúng , sau đó kiểm tra lỗi chính tả và ngữ nghĩa . Thư viện thiết kế là môi trường hỗ trợ cho hệ thống VHDL , tại đây lưu trữ các bản thiết kế đã biên dịch .

Quá trình mô phỏng là mô phỏng một thể được biểu hiện bởi entity-architecture , hoặc configuration hoặc đang đọc bản biên dịch từ thư viện thiết kế và quá trình thực hiện bao gồm các bước sau :

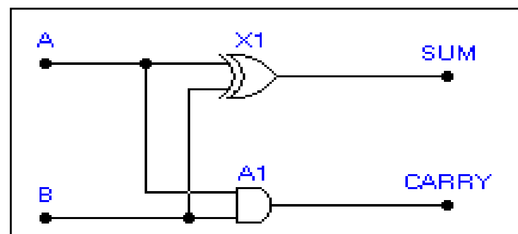
1- Elaboration (chi tiết hoá)
2- Initialization (mã hoá)
3- Simulation (mô phỏng)

2.2 KHAI BÁO THỰC THỂ

Khai báo thực thể là xác định tên của thực thể , tập hợp các cổng giao tiếp . ports là nơi các tín hiêu đi qua , thực thể có thể trao đổi thông tin với các mẫu khác ở môi trường bên ngoài .

Hình 2.3 A Half_Adder circuit

Sau đây là đoạn khai thác thực thể half- adder ởhình f2.3



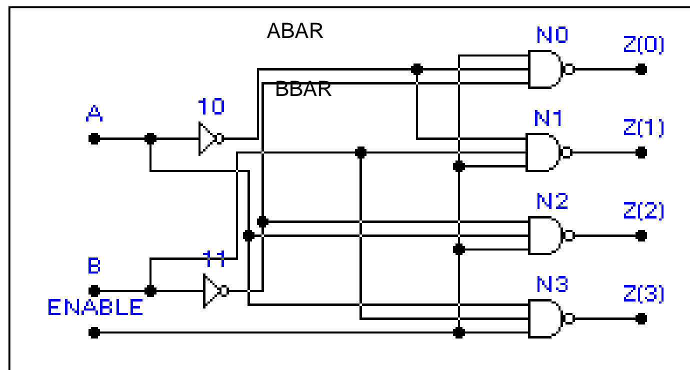

```
entity HALF_ADDER is
      port (A,B:in BIT; SUM, CARRY : out BIT );
end HALF_ADDER;
        -- This is a commet line
```

Thực thể half-adder có hai cổng vào A và B (được xác định là cổng input) và hai cổng ra SUM và CARRY (xác định là cổng output) . Bit là kiểu cơ bản của ngôn ngữ , nó là kiểu đếm , hai loại ký tự : “ 0” và “1” . Kiểu của port được xác định kiểu Bit , giá trị của port là “ 0” hay “ 1”.

Một ví dụ khác về khai báo thực thể vào hai ra bốn , mạch DECODER H2.4

```
entity DECODER2x4 is
      port (A,B,ENABLE : in BIT; Z: out BIT_VECTOR(0 to 3);
```

end DECODER2x4;



Hình 2.4 A 2 to 4 decoder circuit

Thực thể gọi làDECODER 2.4 , 3 cổng vào và 4 cổng ra . BIT-VECTOR là kiểu dãy không giới hạn của BIT . Kiểu dãy không giới hạn là kiểu mà kích thước của dãy không xác định vùng '0 đến 3' xác định kích thước dãy cho part 2

Từ hai ví dụ trên , ta thấy khai báo thực thểkhông xác định được những vấn đề bên trong thực thể , chỉ khai báo tên và các cổng giao tiếp của thực thể .

2.3 THÂN KIẾN TRÚC

Chi tiết bên trong của một thực thể được xác định bởi thân của architecture sử dụng một vài mẫu đặc trưng sau :

1.Tập hợp các kết nối bên trong các thành phần (biểu hiện cấu trúc)
2.Tập hợp các phát biểu gán đồng thời (biểu hiện dòng dữ liệu)
3.Tập hợp các phát biểu gán tuần tự (biểu hiện hành vi)
4.Sự kết hợp các dạng trên

2.3.1 Đặc tả cấu trúc của mô hình

Trong đặc tả cấu trúc của mẫu , một thực thể được mô tả , tập hợp các kết nối bên trong các component (thành phần) mẫu thực thể half-adder (hình 2.3) , được mô tả trong thân architecture sau đây.

```
architecture HA_STRUCTURE of HALF_ADDER is
        component XOR2
                port (X,Y:in BIT; Z: out BIT);
        end coponent;
        component AND2
                port (L,M:in BIT; N: out BIT);
        end coponent;
  begin
         X1: XOR2 port map ( A,B,SUM);
         A1: AND2 port map (A,B,CARRY );
  end HA_STRUTURE;
```

Tên của architecture làHA-STRUCTURE , khai báo thực thể HALF_ADDER (khai báo phần trước) xác định cổng giao tiếpbên ngoài của architecture này . Architecturebao gồm hai phần : Phần khai báo (trước từ begin) và phần phát biểu (sau từ begin) , hai khai báo component được thể hiện ở phần khai báo của architecture . Những phần khai báo này liệt kê các giao tiếp của compenent sử dụng trong

architecture đó . Compenent XOR và AND , cả hai compenent đã có trong thư viện hay chúng hướng đến một compenent khác trong thư viện .

Sự khai báo các thành phần là đối tượng trong phần phát triển của Architecture sử dụng khai báo component instantiation (đối tượng thành phần) . X1 và A1 là các nhãn của component instantiation .

Phần khai báo thứ nhất X1 , chỉ ra tín hiệu A và B (cổng input của HALF_ADDER) , liên hệ tới X và Y là các cổng input của XOR2, cổng output Z của component này liên hệ tới cổng output SUM của entity HALF_ADDER.

Tương tự , trong phần khai báo thứ hai , tín hiệu A và B liên hệ tới cổng L vàM của ADD2 , còn cổng N kiên hệ tới cổng CARRY của HALF_ADDER.

Chú ý trong trường hợp này các tín hiệu trong port map (ánh xạ cổng) của component instantiation và các tín hiệu cổng trong khai báo component phải được đặt đúng vị trí . Mô tả cấu trúc cho HALF_ADDER là chưa nói đến các hàm của nó .

Riêng phần mô tả cho component XOR2 và AND2 , mỗi một thành phần có phần khai báo thực thể và architecture của riêng nó .

Cấu trúc của thực thể DECODER2x4 ở hình 2.4 là :

```
architecture DEC_STR of DECODER2x4 is
        component INV
                port (PIN :in BIT;POUT:out BIT);
        end component;
        component NAND3
                port ( D0,D1,D2 :in BIT;DZ: out BIT);
        end component;
        signal ABAR;BBAR : BIT;
begin
       V0: INV port map (A,ABAR);
       V1: INV portmap (B,BBAR);
       N0: NAND3 port map (ENABLE,ABAR,BBAR,Z(0));
       N1: NAND3 port map (ABAR,B,ENABLE,Z(1));
       N2: NAND3 port map (A,BBAR,ENABLE,Z(2));
       N3: NAND3 port map (A,B,ENABLE,Z(3));
end DEC_STR;
```

Trong vd này tên architecture là DEC_STR, liên kết tới entity DECODER2x4, nó được thừa kế danh sách các cổng giao tiếp từ khai báo entity.

Trong phần architecture , ngoài khai báo 2 component (INV và NAND3) còn có khai báo signal đó là 2 tín hiệu ABAR và BBAR kiểu BIT.

Các tín hiệu này thể hiện dây nối , sử dụng kết nối các component khác nhau từ DECODER. Phạm vi của các tín hiệu này chỉ giới hạn trong architecture đó , có nghĩa là các tín hiệu này không được sử dụng bên ngoài architecture . Ngược lại các tín hiệu của các cổng được khai báo trong thực thể có thể dùng cho bất cứ architecture nào kết nối với thực thể này.

Phát biểu Component instantiation là phát biểu đồng thời , do đó thứ tự các khai báo là không quan trọng. Cấu trúc của kiểu mô tả này chỉ là kết nội trong thành phần (xem như hộp đen), ngoài ý nói về 1 hành vi nào đó của các thành phần thì tự nó cũng không biểu hiện được cái chung củathực thể.

Trong architecture DEC_STR ,tín hiệu A,B vàENABLE sử dụng trong các phát biểu Component instantiation , là các cổng được khai báo trong DECODER2x4.

Ví dụ trong nhãn N3,cổng A liên hệ đến input D0 của component NAND3,cổng B liên hệ đến input D1 , cổng ENABLE liên hệ đến input D2 củaNAND3 , và cổng Z(3) của DECODER2x4 liên hệ đến output DZ củaNAND3. Vị trí các tín hiệu trong port map của Component instantiation với các cổng của component trong phần khai báo phải tương ứng nhau. Hành vi của NAND3 và INV là không rõ ràng trong phần mô tả cấu trúc.

2.3.2 Đặc trưng dòng dữ liệu của mô hình :

Trong đặc trưng của mô hình, dòng dữ liệu qua entity là biểu thức sử dụng các phát biểu gói tín hiệu đồng thời . Cấu trúc của thực thể là kiểu xác định không rõ ràng , chỉ có tính chất suy diễn.

Sau đây xem xét architecture của entity HALF_ADDER :

```
architecture HA_CONCURRENT of HALF_ADDER is
    begin
        SUM <= A xor B after 8 ns;
        CARRY <= A and B after 4 ns;
    end HA_CONCURRENT;
```

Kiểu dòng dữ liệu HALF_ADDER làmô tả sử dụng 2 phát biểu gán tín hiệu đồng thời (phát biểu gán tuần tự được mô tả ở phần sau). Trong 1 phát biểu gán tín hiệu :

```
ký hiệu <=  giá trị của tín hiệu gán vào;
```

Giátrị của biểu thức bên tay phải của phát biểu , qua quá trình tính toán được gán vào tín hiệu phần bên tay trái, gọi là target signal. Phép gán tín hiệu đồng thời được thi hành khi xuất hiện 1 sự kiện tín hiệu của biểu thức bên phải , giá trị của tín hiệu được thay đổi.

Thông tin trễ (delay) được đưa vào phát biểu gán tín hiệu sử dụng mệnh đề after . Nếu 2 tín hiệu A và B là tín hiệu vào của entity HALF_ADDER , sự kiện xuất hiện sau thời gian T, biểu thức bên phải của 2 phép gán được đánh giá .

Tín hiệu SUM được gán giá trị mới sau 8ns, đồng thời tín hiệu CARRY cũng được gán giá trị mới sau 4ns. Khithời gian mô phỏng tới (T+4) ns , CARRY sẽ được gán giá trị mới, cả 2 phát biểu gán được thực hiện đồng thời.

Các phát biểu gán tín hiệu đồng thời là các phát biểu đồng thời,do đó thứ tự các phát biểu trong architecture là không quan trọng , ngoài ra trong architecture HA_CONCURRNT còn có sự liên kết đến khai báo của entity HALF_ADDER.

Đây làdòng dữ liệu củaentity DECODER2x4:

```
architecture DEC_DATAFLOW of DECODER2x4 is
        signal ABAR,BBAR:BIT;
begin
        Z(3) <= not ( A and B and ENABLE);             --statement1
        Z(0) <= not (ABAR and BBAR and ENABLE );   --statement 2
        BBAR <= not B;                                 --statement 3
        Z(2) <= not ( A and BBAR and ENABLE);         --statement 4
        ABAR <= not A;                                 --statement 5
```

```
        Z(1) <= not ( ABAR and B and ENABLE);          --statement 6
    end DEC_DATAFLOW;
```

Thân architecture bao gồm 1 khai báo signal và 6 phát biểu gán đồng thời . Khai báo signal khai báo 2 tín hiệu ABAR và BBAR là biến cục bộ củaarchitecture.

Trong mỗi hành vi của architecture là xem xét có sự kiện 1 tín hiệu input , như là input B tại thời điểm T, thì các phát biểu gán 1,3,6 sẽ đồng thời thực thi. Các biểu thức bên tay phải sẽ được định trị và các giá trị phù hợp sẽ được gán vào các tín hiệu đích tại thời điểm (T + Δ) . Khi thờigian mô phỏng đến (T + Δ) các gía trị mới được gán vào tín hiệu Z(3),ABAR và Z(1). Giá trị của BBAR xuất hiện thì lập tức các phát biểu của 2 và 4 được thực thi , tại thời điểm (T + 2Δ) tín hiệu Z(0) và Z(2) sẽ được gán giá trị mới.

Ngữ nghĩa của hành vi đồng thời này cho biết quá trình mô phỏng đó được phác thảo bằng ngôn ngữ, là sự kiện trigger và thời gian mô phỏng tới thời điểm kế tiếp khi có 1 sự kiện xảy ra.

Ngoài thờigian mô phỏng còn có thể có nhiều thành phần đơn vị thời gian .

Ví dụ các sự kiện đã xảy ra tại các thời điểm 1,3,4,4+Δ,5,6,6+Δ .. đơn vị thời gian.

Mệnh đề after sử dụng chung 1 tín hiệu CLOCK, hãy xem phát biểu gán tín hiệu đồng thời sau :

```
        CLK <= not CLK after 10ns;
```

2.3.3 Đặc trưng hành vi của mô hình :

Một kiểu tương phản với mô tả trước đó là kiểu hành vi của 1 thực thể bao gồm các phát biểu thực hiện liên tục có thứ tự. Tập hợp các phát biểu tuần tự được xác định bên trong phát biểu process. Nó không được xác định trong cấu trúc của thực thể , chỉ là 1 hàm của nó.

Một phát biêu process là 1 phát biểu đồng thời có thể thêm vào architecture.

Ví dụ hành vi cho thực thể DECODER2x4 bao gồm :

```
architecture DEC_SEQUENTIAL of DECODER2x4 is
   begin
     Process (A,B,ENABLE)
             Variable ABAR,BBAR: BIT;
       begin
           ABAR := not A;                               --statement1
           BBAR := not B;                               --statement 2
           If ENABLE = '1' then                         --statement 3
               Z(3) <= not (A and B);                   --statement 4
               Z(0) <= not (ABAR and BBAR) ;            --statement 5
               Z(2) <= not ( A and BBAR);               --statement 6
               Z(1) <= not ( ABAR and B);               --statement 7
           Else
               Z <= "1111";                             --statement 8
           End if;
       End process;
    End DEC_SEQUENTIAL;
```

Một phát biểu process có 1 phần khai báo (trước từ begin) và 1 phần phát biểu (giữa từ begin và end process). Các phát biểu được đưa vào phần phát biểu là các phát biểu tuần tự và sẽ được thực thi 1 cách tuần tự .

Danh sách các tín hiệu trong ngoặc sau từ process thiết lập danh sách 1 cách có thứ tự , phát biểu process thực thi khi có 1 sự kiện trên bất kỳ tín hiệu nào trên danh sách đó .

Trong ví dụ trên ,khi có 1 sự kiện xảy ra trên tín hiệu A,B hoặc ENABLE , các phát biểu trong process sẽ thực thi 1 cách tuần tự.

Khai báo biến (bắt đầu bằng từ variable) khai báo 2 biến ABAR và BBAR, biến khác với tín hiệu là nó luôn được gán giá trị ngay tức khắc và phép gán tín hiệu là := tổ hợp ký hiệu, nó được gán giá trị sau khoảng delay (xác định do người sử dụng hoặc mặc nhiên là khoảng delta) và phép gán điều khiển việc gán giá trị cho biến <= tổ hợp symbol .

Khai báo biến trong process có phạm vi chỉ trong process đó . Biến có thể khai báo trong chương trình con , chương trình con sẽ được bàn luận trong chương 8. Khai báo biến bên ngoài process hoặc chương trình con thì gọi là shared variable. Các biến này có thể cập nhật và đọc nhiều process.

Chú ý tín hiệu không được khai báo trong process . Các phát biểu gán tín hiệu xuất hiện trong process được gọi là phát biêu gán tín hiệu tuần tự , kể cả phát biểu gán biến ,thực hiện tuần tự độc lập với việc xuất hiện các sự kiện trên mỗi tín hiệu trong biểu thức bên tay phải , khác với việc thực thi của các phát biểu gán tín hiệu đồng thời trong phần trước .

Trong architecture , nếu 1 sự kiện xuất hiện trên tín hiệu A,B hoặc ENABLE , phát biểu 1 , phát biểu gán biến thực thixong thì phát biểu 2 mới thực thi và cứ thế
tiếp tục . Với phát biểu thứ 3 , phát biểu if, được điều khiển bởi giá trị của tín hiệu ENABLE, nếu ENABLE lên ‘1’ thì 4 phát biểu gán kế tiếp thực hiện gán giá trị độc lập A,B ABAR,BBAR , tín hiệu đích sẽ được gán cho chúng từng giá trị sau khoảng delay Δ . Nếu ENABLE là ‘0’ thì giá trị ‘1’ được gán cho mọi phần tử output trong dãy Z. Khi thực hiện xong process , process ở trong trạng thái treo và đợi đến khi có sự kiện khác xuất hiện.

Có tồn tại phát biểu case hoặc loop trong process , ngữ nghĩa và cấu trúc của phát biểu này cũng giống như trong các ngôn ngữ cấp cao khác là C hoặc pascal . Có thể sử dụng phát biểu wait trong process. Nó có thể sử dụng để chờ cho 1 giá trị thời gian , cho tới khi điều kiện thành true , hoặc đến khi có 1 sự kiện xuất hiện trên tín hiệu .
Process này không có danh sách nhận vào bởi vì xuất hiện phát biểu wait trong process . Đó là điều quan trọng để nhớ rằng process không bao giờ kết thúc .

Tất cả các process thực hiện suốt quá trình mô phỏng cho đến khi có trạng thái treo.

Sau đây là ví dụ mô tả 1 Flip_Flop :

```
Entity LS_DFF is
        Port ( Q:out BIT; D,CLK : in BIT );
End LS_DFF;

Architecture LS_DFF_BEH of LS_DFF is
        Begin
                Process (D,CLK)
                 Begin
                        If CLK =’1’ then
                                Q <= D;
                        End if  ;
                 End process;
        End LS_DFF_BEH;
```

Process được thực hiện khi có 1 sự kiện trên tín hiệu D hoặc CLK . Nếu CLK =’1’,giá trị của D được gán cho Q . Nếu CLK=’0’ thì không thực hiện phép gán . Trong khi CLK =‘1’ , mọi thay đổi trên D đều xuất hiện trên Q , chỉ khi CLK=’0’ thì giá trị Q vẫn không đổi.

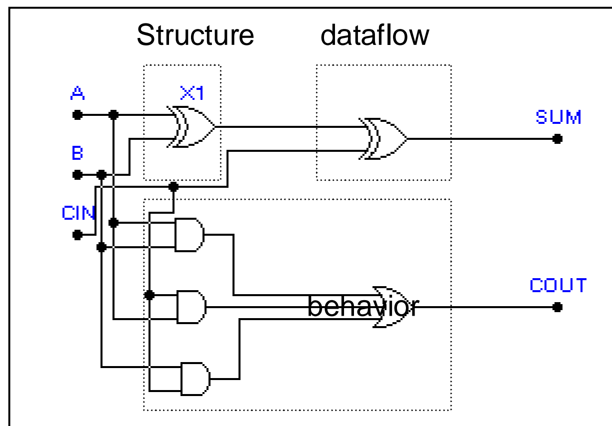
2.3.4 Kiểu hỗn hợp của mô hình :

Có thể hỗn hợp 3 kiểu trên trong 1 architecture. Với architecture này, chúng ta có thể sử dụng các phát biểu component instantiation (đólà biểu hiện structure) ,phát biểu gán tín hiệu đồng thời (biểu hiện dataflow) và phát biểu process (biểu hiện hành vi).

Ví dụ kiểu hỗn hợp cho 1 bit FULL_ADDER xem trong hình 2.7 :

```
Entity FULL_ADDER is
      Port (A,B,CIN :in BIT;SUM,COUT : out BIT);
End FULL_ADDER;

Architecture FA_MIXED of FULL_ADDER is
      Component XOR2
            Port (P1,P2 :in BIT; Pz:out BIT);
      End component;
           Signal S1:BIT;
    Begin
          X1: XOR2 port map (A,B,S1);              --structure
          Process (A,B,CIN)                        --behavior
                Variable T1,T2,T3 :BIT;
          Begin
                T1:=A and B;
                T2:= B and CIN;
                T3:= A and CIN;
                COUN <= T1 or T2 or T3;
          End process;
          SUM <= S1 xor CIN ;                      --dataflow
   End FA_MIXED;
```



Hình2.7 A 1_bit_full_adder

Full_adder sử dụng 1 phát biểu component instantiation , 1 phát biểu process, và 1 phát biểu gán tín hiệu đồng thời . Tất cả các phát biểu này là các phát biểu đồng thời ; do đó thứ tự xuất hiện của chúng trong architecture là không quan trọng . Chú ý rằng bản thân của phát biểu process là 1 phát biểu đồng thời , còn các phát biểu trong process được thực thi 1 cách tuần tự.

S1 làtín hiệu khai báo cục bộ của architecture và được gán giá trị từcổng output của component X1 đến biểu thức gán cho tín hiệu SUM.

2-4 KHAI BÁO ĐỊNH DẠNG (configuration)

Khai báo configuration là sử dụng việc chọn 1 trong những architecture mà entity đãcó và các thành phần bắt buộc, sử dụng mô tả structure trong architecture, tới mô tả thực thểbởi bộ entity _ architecture hoặc bằng 1 configuration, tất cả có trong 1 thư viện thiết kế .

Sau đây là 1 khai báo configuration cho thực thể HALF_ADDE :

```
Library CMOS_LIB, MY_LIB ;
 Configuration HA_BINDING of HALF_ADDER is
       For HA_STRUCTURE
            For X1 : XOR2
                   Use entity CMOS_LIB.XOR_GATE(dataflow);
            End for;
            For A1 : AND2
                   Use configuration MY_LIB.AND_CONFIG;
            End for;
       End for;
 End HA_BINDING;
```

Phát biểu đầu tiên là mệnh đề library với các tên thư viện CMOS_LIB và MY_LIB được khai báo configuration tham khảo đến . tên của configuration là HA_BINDING và xác định dạng cho thực thể HALF_ADDER. Phát biểu kế là xác định thân architecture HA_STRUCTURE (mô tả ở phần 2.3.1) được chọn cho configuration này .

Architecture này gồm 2 component , 2 component bắt buộc phải có .

Phát biểu đầu tiên (for X1: ….. end for), liên quan giữa component instantiation của nhãn X1 tới mô tả thực thể trong bộ entity_architecture . Khai báo thực thể XOR_GATE và architecture dataflow , tất cả có trong thư viện thiết kế CMOS_LIB. Tương tự với component instantiation A1 xuất hiện 1 thực thể xác định bằng khai báo configuration , với tên AND_CONFIG có trong thư viện MY_LIB.

Không có hành vi hoặc ngữ nghĩa mô phỏng kết hợp với khai báo configuration. Nó đơn thuần xác định ràng buộc 1 configuration với 1 entity . Ràng buộc đó được thi hành làm chi tiết quá trình mô phỏng khi thiết kế , và đến khi thiết kế xong thìtổng hợp lại.

Một architecture không có các component instantiation , ví dụ architecture DEC_DATAFLOW có thể chọn thực thể DECODER2x4 để khai báo configuration:

```
Configuration DEC_CONFIG of DECODER2x4 is
       For DEC_DATAFLOW
       End for ;
End DEC_CONFIG;
```

DEC_CONFIG định nghĩa configuration đó là việc chọn architecture DEC_DATAFLOW cho thực thể . Configuration DEC_CONFIG mô tả 1 configuration cho entity DECODER2x4 , có thể mô phỏng tại thời điểm hiện tại.

2.-5 KHAI BÁO GÓI (package) :

Một khai báo pakage sử dụng chứa tập hợp những khai báo chung vào 1 kho , cũng như các component , các type , các procedure, và các function . Các khai báo này có thể đưa vào đơn vị thiết kế khác ,sử dụng mệnh đề *use* .

Sau đây là ví dụ của khai báo pakage :

```
Pakage EXAMBLE_PACK is
      Type SUMMER is ( MAY,JUN,JUL,AUG,SEP );
      Component D_FLIP_FLOP
            Port (D,CK : in BIT ; Q,QBAR : out BIT );
      End component;
      Constant PIN2PIN_DELAY :TIME:= 125 ns ;
      Function INT2BIT_VEC (INT_VALUE : INTEGER )
            Return BIT_VECTOR;
End EXAMBLE_PACK ;
```

Tên của pakage là EXAMBLE_PACK , thân nó bao gồm khai báo type , component ,constant,và function . riêng hành vi của function nó không xuất hiện bên trong khai báo pakage , chỉ xuất hiện phần giao tiếp với function . Phần định nghĩa hoặc phần thân cua function thì xuất hiện trong thân gói (xem phần sau).

Mô phỏng pakage này được biên dịch vào thư viện thiết kế gọi là DESIGN_LIB . Bao gồm các mệnh đề sau có liên quan tới khai báo entity :

```
Library DESIGN_LIB;                      -- this is a library clause.
Use DESIGN_LIB.EXAMBLE_PACK.all;    --this is a use clause.
Entity RX is …
```

Mệnh đề library được tạo từ tên của thư viện DESIGN_LIB mà quá trình mô tả nhìn thấy được , còn có mệnh đề use và quan trọng là khai báo alltrong pakage EXAMBLE_PACK , tất cả được đưa vào khai báo thực thể RX.

Ngoài ra còn có thể chọn các khai báo từ khai báo pakage đưa vào 1 đơn vị thiết kế khác . Ví dụ :

```
Library DESIGN_LIB;
Use DESIGN_LIB.EXAMBLE_PACK.D_FLIP_FLOP;
Use DESIGN_LIB.EXAMBLE_PACK.PIN2PIN_DELAY;
Architecture RX_STRUCTUER of RX is …
```

Hai mệnh đề use tạo từ khai báo component cho D_FLIP_FLOP và hằng PIN2PIN_DELAY mà architecture có thể nhìn thấy được .

Tương tự chọn khai báo từng phần trong pakage bằng các tên chọn . Ví dụ :

```
Library DESIGN_LIB;
Package ANOTHER_PACKAGE is
    Function POCKET_MONEY
            (MONTH :DESIGN_LIB.EXAMBLE_PACK.SUMMER)
             return INTEGER;
    constant TOTAL_ALU: INTEGER ;
end ANOTHER_PACKAGE;
```

Kiểu khai báo SUMMER trong pakage EXAMBLE_PACK là sử dụng trong pakage hiện hành xác định bằng cách chọn tên. Trong trường hợp này , mệnh đề use là không cần thiết . Pakage ANOTHER_PACKAGE gồm 1 khai báo constant với giá trị của nó không xác định , mà nó chỉ mô tả cho deferrend constant. Giá trị của constant thay đổi theo pakage tương ứng .

2.-6 THÂN GÓI (pakage body):

Thân gói sử dụng để chứa những định nghĩa của function và procedure , chúng được khai báo trong pakage tương ứng và khai báo constant cho deferrend constant xuất hiện trong khai báo pakage . Package body luôn gắn liền với khai báo khối , 1 khai báo pakage duy nhất chỉ có 1 pakage body liên kết với nó . Điều này khác với sự liên kết architecture và entity ,nhiều architecture có thể liên kết với 1 khai báo entity . Package body bao gồm các khai báo khác (xem chương 9).

Đây là package body của khai báo package EXAMBLE_PACK trong phần trước :

```
package body EXAMBLE_PACK is
    function INT2BIT_VEC (INT_VALUE: INTEGER)
        return BIT_VECTOR is
    begin
```

```
                Behavior of function described here.
           End INT2BIT_VEC;
       End EXAMBLE_PACK;
```

Tên của package body phải trùng với tên của khai báo package body mà nó liên kết . Đó là điều quan trọng , chú ý nó chỉ không cần thiết nếu như trong khai báo package tương ứng không có các function , procedure và deferred constant .

Sau đây là package body liên kết với gói ANOTHER_PACK trong phần trước :

```
Package body ANOTHER_PACK is
     Constant TOTAL_ALU : INTEGER :=10; --A complete constant
                                          --declaration
     Function POCKET_MONEY                --Function body
        ( MONTH :DESIGN_LIB.EXAMBLE_PACK.SUMMER)
        return INTEGER is
     begin
         case MONTH is
             when MAY => return 5;
             when  JUN|SEP => return 6;       -- when JUN or SEP.
             when  others => return 2;        --when JUN or AUG.
        end case;
    end POCKET_MONEY;
end ANOTHER_PACK;
```

2.-7 KIỂU PHÂN TÍCH (Analysis):

Đều đầu tiên 1 thực thể được mô tả trong VHDL , nó có hiệu lực trong phân tích và mô phỏng . Bước đầu tiên để quá trình có hiệu lực là sự phân tích.

Phân tích tạo 1 file bao gồm 1 hoặc nhiều đơn vị thiết kế (1 đơn vị thiết kế là 1 khai báo entity , 1 architecture , 1 khai báo configuration, 1 khai báo package hoặc 1 package body) và chúng được biên dịch vào trong 1 form . Định dạng này không định nghĩa bằng ngôn ngữ .

Trong suốt quá trình biên dịch ,phân tích làm hiệu lực cú pháp và thực hiện kiểm tra ngữ nghĩa .

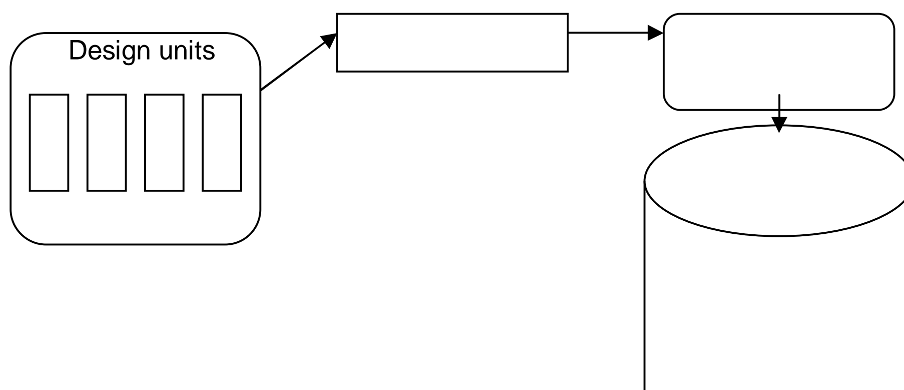
Bộ intermediate form đưa vào trong thư viện thiết kế các thư viện đang làm việc . Thư viện thiết kế là vùng trong môi trường host (máy tính hỗ trợ cho hệ thống VHDL).

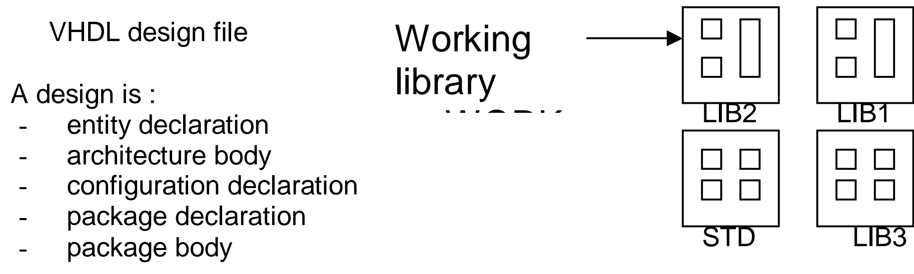
Ở đây mô tả biên dịch là các kho lưu trữ.

Mỗi thư viện thiết kế có 1 cái tên logic sử dụng mô tả cho thư viện trong VHDL . Anh xạ các tên logic này vào vùng lưu trữ vật lý ,là vùng dành riêng bởi môi trường host và không định nghĩa bằng ngôn ngữ.

Ví dụ thư viện thiết kế có thể bổ sung 1 danh bạ trong môi trường host với danh bạ này chứa biên dịch các đơn vị thiết kế.

Sự ánh xạ tên vật lý tới tên logic là sự ánh xạ bên trong file , đó là điều hệ thống VHDL cần phải phiên dịch . Bất kỳ thư viện thiết kế nào cũng có thể tồn tại trong quá trình mô phỏng . Tất cả các thư viện thiết kế có thể cùng tồn tại , 1 phần của thư viện là thiết kế thư viện làm việc với tên logic là work .. Ngôn ngữ thiết kế luôn mô tả biên dịch vào trong thư viện này . Tại 1 thời điểm chỉ 1 thư viện được cập nhật vào . Hình 2.8 chỉ ra quá trình biên dịch :





Hình 2.8 The compilation process.

Thư viện thiết kế với tên gọi là STD được định nghĩa trước trong môi trường VHDL . Thư viện này có 2 package STANDARD và TEXTIO . Package STANDARD gồm những khai báo cho tất cả các kiểu định nghĩa trước của ngôn ngữ (BIT,TIME,INTEGER…), package TEXTIO bao gồm các procedure và các function cần thiết để đọc và viết trong quá trình hoạt động.

Ở đây còn có IEEE package chuẩn ,gọi STD_LOGIC_1164 ,package này định nghĩa 1 nine_value của kiểu logic,gọi là STD_ULOGIC và các kiểu con của nó điều khiển các function và các tiện ích khác . Tiêu chuẩn này gọi là IEEE.STD_1164_1993.

2.-8 MÔ PHỎNG (simulation):

Một mô tả mô hình là đưa kết quả biên dịch vào trong 1 hoặc nhiều thư viện thiết kế , bước kế tiếp trong quá trình thực hiện là mô phỏng.

Kiến trúc thực thể để mô phỏng là tất cả những thành phần ở mức thấp của nó phải được mô tả bằng hành vi.

Sự mô phỏng có thể thực hiện 1 trong 2 phần sau :

+ Một khai báo entity và architecture body.
+ Một configuration.

Quá trình mô phỏng gồm 2 bước lớn :

1. *Elaboration phase :* trong phần này kiến trúc của entity là mở rộng và liên kết , các component hướng đến các entity trong thư viện , và giai đoạn đầu của entity tạo nên mạng hành vi để đọc cho quá trình mô phỏng . Ngoài ra còn cung cấp vùng các tín hiệu , các biến và khai báo hằng trong thiết kế . Giá trị ban đầu gán cho biến và hằng . Các file sẽ được mở nếu nó được chỉ ra trong khai báo của chúng.
2. *Initialization phase :* Kết quả các tín hiệu ảnh hưởng được tính toán , tín hiệu ẩn (bàn luận ở chương sau) được gán trị ,các process được thực thi cho đến khi bị treo và thời gian mô phỏng là 0 ns.

Sự mô phỏng bắt đầu bằng thời gian tăng đến khi có sự kiện kế. Các giá trị đó được gán cho các tín hiệu . Nếu giá trị của tín hiệu thay đổi và nếu tín hiệu đó tồn tại trong danh sách tham khảo của process ,process sẽ thực thi cho tới khi bị treo . Kết thúc mô phỏng khi xuất hiện sự vi phạm của 1 khai báo (assertion), phụ thuộc vào quá trình bổ sung của hệthống VHDL (phát biểu assertion đượcbàn trong chương 4) hoặc khi thời gian quá lớn so với định nghĩa của ngôn ngữ .

CHƯƠNG 3 : PHẦN TỬ NGÔN NGỮ CƠ BẢN

Chương này mô tả những phần tử cơ bản của ngôn ngữ. Những đối tượng dữ liệu đưa vào chứa giá trị của kiểu đã qui định, đúng vậy, chúng mô tả những giá trị không đổi và những yếu tố điều khiển (những yếu tố này điều khiển giá trị dữ liệu)
. Mỗi đối tượng dữ liệu phụ thuộc vào kiểu đặc trưng .Phân loại biến của kiểu và cú pháp cho những kiểu đặc trưng do người sử dụng đặt ra được bàn ở đây .Chương này cũng mô tả phương pháp làm thế nào để kết hợp kiểu với đối tượng bằng cách sử dụng những khai báo đối tượng
Một việc rất quan trọng là hiểu được những lưu ý về kiểu dữ liệu và đối tượng từ khi VHDL là ngôn ngữ kiểu rõ ràng **.(không theo qui tắc:strongly typed language).**Có nghĩa là những toán tử và những phân chia được cho phép trong ngôn ngữ chỉ là kiểu của toán hạng và kết quả tính toán tùy theo những qui tắc ,nó không cho phép đối tượng và literals của những kiểu khác nhau được trộn lẫn tự do trong expresssion
Thí dụ của luật toán tử thêm giá trị real vào 1 giá trị nguyên (integer) và qui 1 giá trị luận lý (boolean) cho một đối tượng kiểu BIT .Vì thế cho nên rất quan trọng để hiểu là kiểu nào và làm thế nào để sử dụng chính xác trong ngôn ngữ
Lần đầu tiên người đọc có thể mong muốn bỏ qua đoạn nói về kiểu ,không đầy đủ kiểu và những tập tin kiểu từ khi những chất liệu là cao cấp hơn .

<u>3.1 IDENTIFIERS:</u>

Có hai loại của identifiers trong VHDL gồm : identifiers cơ bản và identifiers mở rộng .Một identifiers cơ bản trong VHDL bao gồm những phối hợp của 1 hoặc nhiều ký tự. Những ký tự cho phép là một letter hoa (A...Z), một letter thường (a...z), một số digit (0...9) hoặc ký tự gạch dưới(_) .Ký tự đầu tiên trong một indentifier cơ bản phải là 1 letter và ký tự cuối cùng có thể không là ký tự gạch dưới .Chữ hoa và chữ thường được xem như đồng nhất khi sử dụng trong 1 identifier cơ bản , ví dụ như : Count, COUNT, CouNT tất cả đều được xem như nhau trong identifier cơ bản . Tương tự , hai ký tự gạch dưới không thể xuất hiện liên tiếp nhau. Một số ví dụ cho identifiers cơ bản :
DRIVE_BUS SelectSignal RAM_Address
SET_CK_HIGH CONST32_59 r2d2
Một identifier mở rộng là phối hợp của những ký tự được viết giữa hai dấu (\ \).Có thể sử dụng những ký tự bất kỳ ,đưa vào những ký tự như :,!,@,’, and $ .Trong một identifier mở rộng ,ký tự hoa và ký tự thường được phân biệt là khác nhau rõ ràng .Một số ví dụ identifier mở rộng :
\TEST\
\-25\
\2FOR$\
\~Q\
\process\
\~$*****\

\7400TTL\
\----\ \----\
\Count\   khác \COUNT\
Lời chú giải phải được đặt trước bằng hai dấu (-) liên tiếp nhau ,lời chú thích thường đặt ở cuối dòng hoặc đặt tại một vị trí bất kỳ .
Ví dụ :
--this is a comment ; it ends âm tiết the end of this line .
entity UART is end;--This comment starts after the entitydeclaration
Ngôn ngữ xác định một tập từ (words) định trước ,chúng được liệt kê trong chương 1
.Những từ này cũng có thể gọi là từ khóa(keywords),có một ý nghĩa rõ ràng trong ngôn ngữ và do đó không thể sử dụng như những identifier cơ bản

3.2 Đối tượng dữ liệu : (data objects)

Một đối tượng dữ liệu giữ một giá trị của một kiểu dữ liệu rõ ràng. Nó được tạo bởi ý nghĩa của một khai báo đối tượng.Ví dụ như :
variable COUNT : INTEGER ;
Kết quả này trong phần tạo ra của một đối tượng dữ liệu gọi là COUNT, nó có thể giữ giá trị nguyên. Đối tượng COUNT cũng được khai báo là lớp biến (variable class)
Mỗi một đối tượng dữ liệu phụ thuộc vào một trong bốn lớp sau :

1. Constant : một đối tượng của lớp constant (thường được gọi là constant) có thể
giữ một giá trị duy nhất của kiểu đã cho. Giá trị này được gán cho một constant trước khi bắt đầu , và giá trị không thể thay đổi trong suốt **trường hợp**. để khai báo một constant trong một subprogram, giá trị sẽ được gán cho constant ở mỗi thời điểm chương trình con (subprogram) được gọi
2. Variable (biến) : một lớp đối tượng biến (thường gọi là variable) có thể giữ một giá trị duy nhất của kiểu đã cho .Tuy nhiện trong trường hợp này giá trị khác nhau có thể được gán vào biến tại những thời điểm khác nhau (sử dụng biểu thức gán biến)
3. Signal (tín hiệu) : Một đối tượng tùy thuộc vào lớp tín hiệu thường gọi là signal ,giữ 1 loạt các giá trị ,những giá trị này bao gồm giá trị hiện hành của tín hiệu và 1 tập hợp các giá trị tương lai có thể có(xuất hiện trong tín hiệu) .Những giá trị tương lai có thể được gán cho 1tín hiệu (sử dụng cho một biểu thức gán tín hiệu)
4. File (tập tin) :1 đối tượng tùy thuộc vào lớp tập tin (thường gọi là file).Bao gồm những giá trị liên tiếp nhau .Những giá trị có thể được đặt hoặc được viết vào 1 file sử dụng cho những chương trình con đọc và những chương trình con viết tương ứng .

   Tín hiệu có thể được coi như một dây trong mạch ,khi những biến và hằng giống nhau ở những biến đếm trong những ngôn ngữ lập trình bậc cao như C hoặc PASCAL .Tín hiệu được coi như nhau (sử dụng những kiểu thiết kế dây và FLIPFLOP.Khi những biến và hằng này được sử dụng để thiết kế hành vi mạch điện .1file được sử dụng như một kiểu mẫu tập tin trong môi trường chính .

Một khai báo đối tượng được sử dụng cho việc khai báo đối tượng ,kiểu của nó và lớp của nó .Một giá trị có thể được gán tùy ý cho một tín hiệu ,1 biến hoặc 1 hằng .Đối với 1 file ,một khai báo đối tượng có thể có thông tin rõ ràng trong việc làm thế nào để mở 1 file

CONSTANT DECLARATIONS :

Những thí dụ khai báo hằng như sau :
constant RISE_TIME:TIME:=10 ns
constant BUS_WIDTH:INTEGER:=8;
Khaibáo đầu tiên là khai báo đối tượng RISE_TIME nó có thể chứa giá trị của kiểu thời gian (một kiểu được khai báo trước trong ngôn ngữ)và giá trị được gán vào đối tượng tại thời điểm là 10 ns .Biến hằng thứ hai được khai báo là BUS_WIDTH,với kiểu số nguyên ,giá trị là 8
Một thí dụ khác minh họa cho việc khai báo hằng là :
constant NO_OF_INOUTS:INTEGER;

Giá tri hằng không được rõ ràng trong trường hợp này .Như vậy nó được gọi là một hằng trì hoãn .Nó chỉ có thể xuất hiện trong khai báo gói .một khai báo hằng đầy đủ với giá trị kết hợp phải xuất hiện trong phần thân gói tương ứng

VARIABLE DECLARATIONS :

Một số thí dụ khai báo biến là :

```
variable CTRL_STATUS :BIT_VECTOR (10 downto 0);
variable SUM :INTEGER range 0 to 100 :=10;
variable FOUND,DONE:BOOLEAN;
```

Khai báo đầu tiên chỉ rõ là một đối tượng biến CTRL_STATUS là một dãy có 11 phần tử ,với mỗi phần tử trong dãy có kiểu là BIT.Trong khai báo thứ hai giá trị đầu thể hiện được gán vào biến SUM,Khi trường hợp bắt đầu ,SUM sẽ có giá trị khởi đầu là 10 .Nếu không có giá trị đầu rõ ràng cho mọi biến .Giá trị mặc định sẽ được sử dụng như một giá trị khởi đầu .Giá trị mặc định này là T'LEFT.Khi T là đối tượng và 'LEFT là thuộc tính được xác định trước của kiểu đã được đặt ra cho giá trị cực trái trong tập hợp giá trị phụ thuộc có kiểu T .Trong khai báo thứ ba,giá trị khởi đầu sẽ được gán cho FOUND và DONE trong trường hợp giá trị hỡi đầu là sai (sai là giá trị cực trái của kiểu BOOLEAN xác định trước).Nếu kiểu của một biến là kiểu dãy hoặc 1 kiểu record giá trị khởi đầu của mỗi phần tử trong dãy của CTRL_STATUS là '0'

SIGNAL DECLARATIONS

sau đây là những thí dụ của khai báo tín hiệu :

```
sihnal CLOCK:BIT;
signal DATA_BUS :BIT_VECTOR (0 to 7);
signal GATE_DELAY:TIME:=10 ns;
```

Việc giải thích của những khai báo tín hiệu thì rất giống nhau đối với khai báo biến .Khai báo tín hiệu đầu tiên là khai báo tín hiệu CLOCK của kiểu BIT và lấy giá trị đầu là '0' (0 là giá trị cực trái của kiểu BIT).Khai báo tín hiệu thứ ba là khai báo đối tượng tín hiệu GATE_DELAYcủa kiểu thời gian có giá trị đầu là 10 ns

FILE DECLARATION:

Một file được khai báo sử dụng khai báo file với cú pháp khai báo file là :

```
file file_nam :file-type-name[[open mode ] is string-expression];
```

chuỗi biểu thức được làm sáng tỏ bởi môi trường chính như tên vật lý của file .Kiểu ghi chú trong trường hợp tập tin đã được sử dụng như kiểu chỉ được đọc hoặc chỉ được viết ,hoặc trong kiểu nối. Sau đây là một số thí dụ khai báo file :

```
Type STD_LOGIC_FILE is file of STD_LOGIC_VECTOR;
Type BIT_FILE is file of BIT_VECTOR;
file STIMULUS :TEXT open READ_MODE is "usr /home/james/add.vec";
file PAT1,PAT2:STD_LOGIC_FILE
```

OTHER WAYS TO DECLARE OBJECTS:

Không có đối tượng nào trong một mô tả VHDL được tạo ra rõ ràng để sử dụng khai báo đối tượng .Những đối tượng khác như một trong những đối tượng sau :

1. những cổng của thực thể .tất cả cổng là đối tượng tín hiệu
2. generic của thực thể ,chúng là đối tượng không đổi
3. Thông số hình thức của hàm và thủ tục .Thông số hàm là hằng hoặc tín hiệu ,thông số thủ tục có thể phụ thuộc vào bất kỳ lớp đối tượng nào .

Thí dụ sau minh họa cho khai báo ẩn trong một phát biểu lặp FOR :

```
for COUNT in 1 to 10 loop
        SUM:=SUM+COUNT;
end loop;
```

Trong phát biểu lặp FOR này ,COUNT là một hằng được khai báo ẩn của kiểu integer trong khảng từ 1 đến 10 .Vì thế không thể khai báo rõ ràng .Hằng COUNT được tạo ra khi vào vòng lặp đầu tiên và không còn tồn tại khi thoát khỏi vòng lặp

3.3 DATA TYPES :

Mỗi đối tượng dữ liệu trong VHDL có thể giữ 1 giá trị phụ thuộc vào tập giá trị được xác định bởi việc sử dụng một khai báo kiểu .Một kiểu là 1 tên kết hợp với 1 tập giá trị và một tập tác vụ .Những kiểu nào đó và những tác vụ được trình bày trong đối tượng của kiểu đó thì được định nghĩa trước trong ngôn ngữ .Thí dụ kiểu integer là 1 kiểu không xác định với tập giá trị là những số nguyên trong vùng xác định được cung cấp bởi hệ thống VHDL .Vùng tối thiểu phải được cung cấp là $-(2^{31}-1)$ đến $(2^{31}-1)$.Các tác vụ thương xuyên sử dụng là :

+,-,*,/ và boolean co hai giá trị là true và false cùng với những tác vụ OR,AND,NOR... và NOT.Khai báo cho 1 kiểu được định nghĩa trước được khai báo trong gói chuẩn standar.Những toán tử cho những kiểu này được định nghĩa trước trong ngôn ngữ .Ngoài ra còn những kiểu khác có thể tồn tại trong ngôn ngữ được phân thành 4 loại sau :

1. Scalar type (kiểu vô hướng):những giá trị phụ thuộc vào những kiểu xuất hiện theo thứ tự liên tục
2. Kiểu composite (đa hợp):gồm những phần tử kiểu đơn lẻ,kiểu dãy hoặc những phần tử kiểu khác nhau
3. Kiểu access Cung cấp thêm vào đối tượng đã cho theo đường con trỏ
4. Kiểu file : Cung cấp thêm vào đối tượng chứa đựng một dãy giá trị của kiểu đã cho

Có thể nhận được những kiểu con từ những kiểu được định nghĩa trước do người sử dụng định nghĩa.

3.3.1.SUBTYPE :Kiểu con là 1 kiểu với điều kiện xác định một tập giá trị con cho một kiểu con.một đối tượng được gọi là kiểu cơ sở .Một kiểu được gọi là kiểu con nếu nó phụ thuộc kiểu cơ sở và thỏa mãn điều kiện khai báo kiểu con

Thí dụ :

```
Subtype my-integer is integer rang 48 to 156
type digit is (‘0’,’1’,’2,’3’,’4,’5,’6,’7,’8,’9’);
subtype MIDDLE is DIGIT range ‘3’ to’7’;
```

3.3.2 SCALAR TYPE:

Những giá trị của kiểu này được sắp xếp có nghĩa là tác vụ quan hệ có thể được sử dụng cho những giá trị này .Thí dụ

BIT là một kiểu scalar type và biểu thức ‘0’<’1’ là đúng và có giá trị là true .Có 4 loại kiểu vô hướng khác nhau đó là :

1. enumeration
2. integer
3. physical
4. floating point

Xét mỗi kiểu :

1. **Enumeration** (Kiểu liệt kê) :Một khai báo kiểu liệt kê định nghĩa một kiểu tập hợp những giá trị được người sử dụng định nghĩa bao gồm những đặc tính và những ký tự .Thí dụ :

type MVL is (‘U’,’0’,’1’,’Z’)

Trật tự các giá trị xuất hiện trong khai báo kiểu liệt kê định nghĩa thứ tự của chúng .Có nghĩa là khi sử dụng các toán tử quan hệ 1 giá trị luôn ít hơn giá trị bên phải của nó trong trật tự

2. **Integer** : Được định nghĩa là 1 kiểu tập hợp những giá trị rơi vào vùng số nguyên được chỉ ra .Thí dụ

```
TYPE index is range 0 to 15;
type  WORD_LENGTH is range 31 downto 0;
```

Những giá trị kiểu nguyên được gọi là tham số nguyên

3.Floating point :

Kiểu floating point có một tập giá trị trong vùng số thực .Thí dụ

Type TTL_VOLTAGE is range -5.5 to -1.4;

type REAL_DATA is range 0.0 to 31.9
Giới hạn điều kiện xác định trong 1 khai báo kiểu floating point phải là cố định hoặc những biểu thức tính cục bộ .
floating point literal là giá trị của kiểu floating point .Một vài thí dụ của floating point literals là:
16.26 0.0 0.002 3_1.4_2
floating point literals khác nhau từ literals số nguyên (integer literals) bởi sự có mặt của dấu (.) (the dot), dù vậy 0 là một literal nguyên khi 0.0 là một floating point literal .
floating point literals cũng có thể được trình bày theo một kiểu số mũ.Số mũ được trình bày thành một luỹ thừa của 10 và giá trị số mũ phải là một số nguyên .Ví dụ như :
62.3E-2 50.E+2
số nguyên và floating point literals cũng có thể được viết trên nền khác hệ thập phân (decimal).Nền có thể là bất kỳ giá trị nào từ 2 đến 16 .Do đó literals có thể gọi là based literals .Trong trường hợp này ,số mũ biểu diễn thành 1 lũy thừa trên một nền được giải thích .Cú pháp cho một based literals là :

```
base# based _value#              --form 1
base# based _value#E exponent    --form 2
```

Một số ví dụ :
2#101_101_000# biểu diễn cho $(101101000)_2$ =(360) trong hệ thập phân
16#FA# biểu diễn cho $(FA)_{16}$ = $(11111010)_2$ = (250) trong hệ thập phân
16#E#E1 biểu diễn cho $(E)_{16}$ $*(16^1)$=14*16=(224) trong hệ thập phân
2#110.01# biểu diễn cho $(110.01)_2$ =(6.25) trong hệ thập phân
Nền và giá trị số mũ trên 1 based literal phải ở trong một ký hiệu thập phân
Chỉ có duy nhất kiểu floating point là REAL .Vùng của REAL là sự lệ thuộc vào bổ sung một lần nữa ,nhưng ít nhất phải trải trong vùng giới hạn từ -1.0E38 đến +1.0E38,và phải độ chính xác ít nhất 6 số thập phân .

4.Physical types

Một kiểu physical chứa đựng những giá trị tượng trưng cho kích thước của một vài tiêu chuẩn đo lường chất lượng ,như thời gian ,điện áp, luồng điện .Giá trị của kiểu này biểu diễn như một bội số nguyên của 1 base unit .Ví dụ của một khai báo kiểu physical là :

```
Type CURRENT is rang 0 to 1E9
units
nA;                -- nano_ampere
uA     =1000nA; --micro_ampere
mA     =1000uA;-- milli_ampere
Amp    =1000mA; --ampere
end units;
Subtyoe FILTER_CURRENT  is  CURRENT range 10 uA to 5 mA;
```

CURRENT xác định cho 1 kiểu physical ,nó chứa đựng giá trị từ 0 nA cho đến 10^9nA.nền unit là một nano_ampere ,khi tất cả những cái khác đều lấy từ units.Vị trí số của gía trị là số của nền units tượng trưng bởi giá trị này .Ví dụ :
2uA có một vị trí 2000 khi 100nA có vị trí 100.Vùng của giá trị có thể bao gồm cả những giá trị âm.
Ví dụ :

```
Type STEP_TYPE is range -10 to +10
  units
        STEP;                --base unit
        STEP2=2STEP;--chuyển hóa từ unit
        STEP5=5STEP;--chuyển hóa từ unit
  end units;
```

Giá trị trong kiểu này từ -10 STEP cho đến +10 STEP .Những unit khác của kiểu này là STEP2 và STEP5

Giá trị của kiểu physical(vật lý) được gọi là Physical literals. Physical literals được viết như một integer hoặc 1 floating point literaltheo sau bởi tên unit.Ví dụ :"10 nA" là literal physical(lưu ý khoảng giữa 10 và nA là cần thiết),khi "Amp" cũng là 1 literal ,nó ngụ ý là 1 Amp .Những ví dụ khác là :
100 ns
10 V
50 sec
Kohm xem như 1Kohm
5.2 mA tương đương 5200uA
2.5 STEP2 tương đương 5 STEP
5.6 nA là 5nA
7.2 STEP là 7 STEP (phần phân số đã được làm tròn)
Kiểu vật lý chỉ được khai báo trước là TIME,và nằm trong vùng giá trị nền (base unit).Vùng này lệ thuộc vào bổ sung một lần nữa ,nhưng ít nhất phải trải trong vùng giới hạn từ -(2^{31}-1) đến +(2^{31}-1).Đây cũng là một kiểu subtype physical khai báo trước .Khai báo của kiểu TIME và kiểu con DELAY_LENGHT xuất hiện trong gói chuẩn (STANDARD)

3.3.3 COMPOSITE TYPES :

Kiểu đa hợp tượng trưng cho sự tập hợp các giá trị .Có hai kiểu đa hợp :kiểu dãy (ararraytype) và kiểu record (record type) . Một kiểu array đại diện cho một tập hợp của các giá trị phụ thuộc vào một kiểu duy nhất, mặt khác ,kiểu record đại diện cho một tập các giá trị phụ thuộc vào các kiểu khác nhau .Một đối tượng phụ thuộc vào kiểu đa hợp cho nên đại diện cho một tập hợp của các đối tượng con (subobjects),một trong mỗi phần tử của kiểu đa hợp.Kiểu đa hợp có thể có giá trị phụ thuộc vào kiểu vô hướng (scalar type), một kiểu đa hợp, hoặc một kiểu thêm vào (access type)

ARRAY TYPES:

Một đối tượng kiểu dãy bao gồm những phần tử có cùng một kiểu .Ví dụ một khai báo dãy như sau :

```
type ADDRESS_WORD is array (0 to 63) of BIT;
type DATA_WORD is array (7 downto 0) of MVL;
type ROM is array (0 to 125) of DATA_WORD;
type DECODE_MATRIX is array (POSITIVE range 15 downto 1, NATURAL range 3 downto 0 ) of MVL;
subtype NATURAL is INTEGER range 0 to INTEGER'HIGHT;
subtype POSITIVE is INTEGER range 0 to INTEGER'HIGHT ;
```

Ví dụ những kiểu khai báo sử dụng những kiểu trên là :

```
    variable ROM_ADDR:ROM;
    signal   ADDRESS_BUS:ADDRESS_WORD;
    constant DECODER:DECODE_MATRIX;
    variable DECODE_VALUE:DECODE_MATRIX;
```

ADDRESS_BUS là một đối tượng dãy thứ nguyên (one_dimensional)nó bao gồm 64 phần tử có kiểu BIT , ROM_ADDR là một đối tượng dãy thứ nguyên (one_dimensional)nó bao gồm 126 phần tử mỗi phần tử là đối tượng dãy khác chứa đựng 8 phần tử của kiểu MVL .Vậy ta đã có dãy của dãy .

Những phần tử của dãy có thể được gia tăng bởi việc liệt kê giá trị trong một dãy .Thí dụ :ADDRESS_BUS(26) chuyển đến phần tử thứ 27 của đối tượng dãy ADDRESS_BUS.ROM_ADDR(10)(5) chuyển đến giá trị (của kiểu MVL) tại index 5 của đối tượng dữ liệu ROM_ADDR(10) (của kiểu DATA_WORD),DECODER(5,2) chuyển đến giá trị của phần tử tại cột thứ hai và hàng thứ năm của đối tượng 2 dãy thứ nguyên (two_dimensional) .Lưu ý sự khác nhau trong địa chỉ của 1 kiểu dãy two_dimensionalvà một dãy của một kiểu là một dãy của một kiểu khác .

Ngôn ngữ cho phép một số tùy ý với kích cỡ có thể hợp với một dãy .Nó cũng cho phép một đối tượng dãy có thể được gán cho một đối tượng dãy khác có cùng kiểu có bởi do sử dụng một phát biểu gán .Phép gán có thể tạo ra một dãy nguyên vẹn .hoặc một phần của dãy .

Thí dụ :

```
ROM_ADDR(5):="01000100"      --gán cho một phần tử của một dãy
DECODE_VALUE:=DECODER; --một kiểu dãy nguyên vẹn đã được gán
ADDRESS_BUS(8 to 15)<=X"FF" -- gán cho một phần của một dãy .
```

Những thí dụ của các kiểu dãy trên là bắt buộc cho những khai báo dãy ,thì số lượng của những phần tử trong kiểu là rõ ràng .Ngôn ngữ cũng cho phép những kiểu dãy có thể không giới hạn .Trong trường hợp này số phần tử trong dãy thì không rõ ràng trong khai báo kiểu .Lẽ ra ,một khai báo đối tượng cho một đối tượng mà kiểu đó khai báo số phần tử của dãy .Một ràng buộc khai báo subtype kiểu dãy cũng có thể rõ ràng do việc ràng buộc liệt kê cho một kiểu dãy không ràng buộc .Một tham số chương trình con có thể là một kiểu không ràng buộc .trong trường hợp này ràng buộc chứa đựng từ những tham số thực qua trong suốt lần gọi chương trình con (chương trình con được bàn tới ở chương 8) .Thí dụ của khai báo dãy không ràng buộc là :

```
Type STACK_TYPE is array (INTEGER rang <>)
        of ADDRESS_WORD;
subtype STACK is STACK_TYPE (0 to 63);
type OP_TYPE  is (ADD,SUB,MUL,DIV);
type TIMING is array(OP_TYPE range <>,OP_TYPE range <>)
of TIME;
```

Những thí dụ của việc khai báo của những kiểu đối tượng trên là :

```
variable FAST_STK :STACK_TYPE ( -127 to 127);
constant ALU_TIMING:TIMING:=--ADD,SUB,MUL
((10 ns, 20 ns, 45 ns),                    --ADD
(20 ns, 15 ns, 40 ns),                     --SUB
(45 ns, 40 ns, 30 ns))                     --MUL
```

STACK_TYPE được xác định là một kiểu dãy không ràng buộc được làm rõ bởi phần liệt kê của dãy như một kiểu integer,và kiểu phần tử như kiểu ADDRESS_WORD .STACK là một kiểu con của kiểu nền STACK_TYPE với phần ràng buộc được liệt kê thứ tự .khai báo biến cho FAST_STK xác định bởi kiểu STACK_TYPE , cũng được làm rõ bởi ràng buộc liệt kê.Hằng ALU_TIMING định rõ thời gian cho 2 toán tử ALU,khi toán tử có thể là ADD,SUB hoặc MUL Thí dụ một ALU có thể đóng vai trò toán tử ADD và SUB có trì hoãn 20 ns .Khai báo cho ALU_TIMING là trường hợp đặc biệt của khai báo hằng .Khi không ràng buộc cần chỉ rõ kiểu dãy không ràng buộc vì thế kiểu của đối tượng hằng được xác định từ số của những giá trị trong hằng .

Có hai kiểu dãy không ràng buộc thứ nguyên được xác định trước trong ngôn ngữ .STRING và BIT_VECTOR .STRING là một kiểu dãy của tự ,trong khi BIT_VECTOR là kiểu dãy của BIT.

Ví dụ như là :

```
variable MESSAGE:STRING(1 to 17) :="hello,VHDL World";
signal RX_BUS :BIT_VECTOR (0 to 5) :=O"37';
--O"37" là một chuỗi thực xuất hiện lại ở giá trị 37
constant ADD_CODE :BIT_VECTOR :=('0','1','1','1','0');
```

Một giá trị đại diện cho một dãy thứ nguyên của ký tự được gọi là *string literal. string literals* được viết bởi dãy ký tự trong dấu nháy đôi .

Những ví dụ *string literals* là :

```
"THIS IS A TEST"
"SPIKE DETECTED ! "
"State ""READY"" entered !"
```

Một *string literal* có thể được gán cho những kiểu khác nhau của một đối tượng ,ví dụ ,cho một đối tượng kiểu STRING hoặc một kiểu đối tượng BIT_VECTOR .Kiểu của một *string literal* được xác định từ phạm vi nơi chúng xuất hiện .Sau đây là những ví dụ :

---example 1 :

```
variable ERROR_MESSAGE :STRING(1 to 19);
ERROR_MESSAGE:="fatal ERROR :abort !";
```

---example 2 :

```
variable BUS_VALUE :BIT_VECTOR (0 TO 3);
BUS_VALUE:="1101";
```

.Trong ví dụ đầu tiên , *string literal* là kiểu STRING ,trong khi đó ,trong thí dụ 2 ,*string literal* có kiểu BIT_VECTOR .Kiểu của một *string literal* cũng có thể có trạng thái rõ ràng bởi sử dụng biểu thức định lượng (qualified expression)

Một *string literal* đại diện cho một loạt BIT liên tục (giá trị của kiểu BIT),cũng có thể được xuất hiện như một bit *string literal* .Những BIT liên tục này gọi là *bit strings* ,có thể được thể hiện lại như một giá trị nhị phân ,một giá trị octal ,hoặc một giá trị hexa decimal.Ký tự gạch dưới có thể tự do sử dụng trong một *bit string literal* cho trong sáng dễ hiểu.Ví dụ như:

```
X"FFO"              --X for hexa decimal .
B"00_0011_1101"     --B for binary .
O"327"          --O for octal .
```

kiểu của một *bit string literal* cũng có thể được xác định từ phạm vi nơi nó xuất hiện .Ví dụ như :

```
Type MVL is ('X','0','1','Z');
type MVL_vector is array (NATURAL range <>) of MVL;
variable FXA:MVL_VECTOR (0 to 3);
variableBRY:BIT_VECTOR(7 downto 0);
. . .
FXA:=B"01_00";
BRY:=X"AB";
```

Một chuỗi bit thực sự trong phát biểu gán đầu tiên là kiểu MVL_VECTOR,trong khi chuỗi bit thực trong phát biểu gán thứ hai là kiểu BIT_VECTOR.

Có rất nhiều điểm khác nhau trong việc gán giá trị cho một đối tượng dãy.Sau đây là những thí dụ :

varibale OP_CODES:bit_vector(1 to 5);

OP_CODES:="01001"; --Một chuỗi được gán

OP_CODES:=('0','1','0','0','1') --Giá trị đầu được gán cho OP_CODES(1),giá trị thứ hai được gán cho OP_CODES(2),và cứ thế .

OP_CODES:=(2=>'1', 5=>'1', others=>'0')--phần tử thứ hai và thứ năm của OP_CODES được gán cho giá trị '1' và những giá trị còn lại được gán cho '0'.

OP_CODES:=(others=>'0'); --tất cả các giá trị được đặt về 0.

Lưu ý khi sử dụng những từ khóa khác có nghĩa chỉ những giá trị không được gán trước đó ,khi sử dụng ,chúng phải là kết hợp cuối cùng .Biểu thức sử dụng 3 phép gán cuối cùng cho OP_CODES là những thí dụ của kiểu dãy tổng hợp(array aggregate) .Một aggregate là tập hợp những phần tử cách nhau bởi dấu phẩy riêng lẻ (comma_separated)được đóng mà không có ngoặc đơn (parenthesis).

KIỂU RECORD:

Một đối tượng của kiểu record bao gồm những phần tử giống nhau hoặc khác nhau kiểu .Một kiểu record thì tương đương với kiểu dữ liệu record trong PASCAL và khai báo kết cấu trong C.Thí dụ kiểu khai báo record như sau :

```
typePIN_TYPE is  range 0 to 10;
typeMODULE is
record
        SIZE:INTEGER range 20 to 200;
```

```
                CRITICAL_DLY:TIME
                NO_INPUTS:PIN_TYPE;
                NO_OUTPUT:PINTYPE ;
        end record;
```

Giá trị có thể được gán cho những đối tượng record sử dụng tổng hợp .Thí dụ như :

```
    variable NAND_COMP:MODULE;
        NAND_COMP là một đối tượng kiểu RECORD kiểu MODULE
        NAND_COMP:=(50,20 ns,3 ,2);
```

--Ngụ ý 50 được gán cho size ,20 ns được gán cho CRITIAL_DLY,v.v

Giá trị được gán cho một đối tượng record từ một đối tượng record khác có cùng kiểu sử dụng biểu thức gán riêng lẻ .Trong thí dụ sau mỗi phần tử củaNAND_GENERIC được gán cho giá trị của phần tử tương ứng trong

-NAND_COMP.

```
Signal NAND_GENERIC : MODULE ;
NAND_GENERIC<=NAND_COMP;
```

mỗi phần tử của đối tượng record củng có thể được gán riêng lẻ bởi việc sử dụng những tên lựa chọn .

Thí dụ :

```
NAND_COMP.NO_INPUT:=2;
```

Giá trị tổng hợp có thể được gán cho những đối tượng record (sử dụng cả hai :vị trí và tên kết hợp).Do đó một kiểu của tổng hợp được xác định từ phạm vi mà nó sử dụng .Tổng hợp có thể là một dãy hoặc 1 record tổng hợp ,tùy thuộc vào cách sử dụng .

Thí dụ :

```
CAB:=(others=>'0');
```

Nếu CAB là một đối tượng kiểu dãy ,tổng hợp được đối xử như một dãy tổng hợp .Nói một cách khác .Nếu CAB là một đối tượng kiểu record tổng hợp là một record tổng hợp ở nơi những phần tử khác được gán cho tất cả các phần tử trong record .

3.3.4 ACCESS TYPES

Giá trị tùy thuộc 1 kiểu access là những pointer đến đối tượng được chỉ ra của những kiểu khác .Nó tương tự như những pointer trong ngôn ngữ PASCAL và ngôn ngữ C.Những thí dụ của khai báo kiểu access như sau :

--MODULE là kiểu khai báo record trong đoạn sau :

```
    type PRT is access  MODULE;
    type FIFO is array(0 to 63 ,0 to 7) of BIT;
    type FIFO_PTR is access FIFO;
```

PTR là một kiểu access ,những giá trị của nó là những địa chỉ vạch ra đối tượng của kiểu MODULE .Mỗi một kiểu access cũng có thể có giá trị rỗng ,có nghĩa là nó chưa được chỉ đến bất kỳ đối tượng nào .Những đối tượng của một kiểu có thể chỉ tùy thuộc vào lớp biến .khi một đối tượng của một kiểu access được khai báo .Giá trị mặc định của đối tượng đó là NULL .

Thí dụ :

```
variable MOD1PTR,MOD2PTR:PTR—Giá trị mặc định là NULL
```

Đối tượng mà kiểu accesss chỉ đến có thể được tạo bằng cách sử dụng *allocators* , *allocators* cung cấp 1 cơ cấu để tạo ra một đối tượng như một kiểu rõ ràng

```
MOD1PTR:=new MODULE;
```

new trong phép gán là nguyên nhân đối tượng của kiểu MODULE đã được tạo ,và chỉ đến đối tượng trả về .Những giá trị của những phần tử của record MODULE là những giá trị mặc định của mỗi phần tử .Đó là 20 kích cỡ mỗi phần tử ,TIME'LEFT cho phần tử CRITACAL_DLY, và giá trị 0(đây là PIN_TYPE'LEFT) cho những phần tử NO_INPUT va NO_OUTPUT .Những giá trị đầu cũng có thể được gán cho 1 đối tượng được tạo mới bởi những giá trị rõ ràng .Điều này được tế hiện trong thí dụ sau :

MOD2PTR:=new MODUKE'(25,10ns,4,9);

Những đối tượng của 1 kiểu access có thể được tham khảo như:

1. ***obj-ptr.all*** **:** Quyền lui tới toàn thể đối tượng được chỉ bởi obj_ptr nơi mà obj-ptr là một pointer chỉ đến một đối tượng của bất kỳ kiểu nào .

2.array-obj-ptr*(element-index)* :quyền lui tới của dãy phần tử rõ ràng nơi mà array-obj-ptr là một pointer chỉ đến một đối tượng dãy .

3.record-obj-ptr.element-name : :quyền lui tới của 1 phần tử record rõ ràng nơi mà record-obj-ptr là một pointer chỉ đến một đối tượng record .

Những phần tử của đối tượng mà MOD2PTR chỉ đến có thể được thêm vào như :MOD2PTR.SIZE, MOD2PTR.CRITICAL_DLY, MOD2PTR.NO_INPUTS and MOD2PTR.NO_OUTPUTS,cung cấp pointer là NOT NULL.

Đối với mỗi kiểu access ,thủ tục DEALLOCATE đã được khai báo ngầm ,thủ tục này ,khi đã gọi trả về nơi lưu trữ bị chiếm giữ bởi đối tượng trong môi trường chính .Đối với kiểu PTR va FIFO_TR khai báo sớm hơn ,thủ tục DEALLOCATE theo sau được hiểu ngầm như sau :

procedure DEALLOCATE (P : inout PTR) ;
procedure DEALLOCATE (P : inout FIFO_PTR) ;

Phần thực thi của phát biểu sau : DEALLOCATE (MOD2PTR);
là nguyên nhân nơi lưu trữ bị chiếm giữ bởi đối tượng mà MOD2PTR chỉ đến ,bị deallocated ,và MOD2PTR chuyển thành NULL.

Pointers có thể được gán cho một biến pointer khác của kiểu access tương tự ,như là : MOD1PTR:=MOD2PTR;
là luật gán .Bây giờ cả hai MOD1PTR và MOD2PTR chỉ đến cùng một đối tượng .Xem lại thí dụ sau :

type BITVEC_PTR is access BIT_VECTOR ;
variable BITVEC:BITVEC_PTR :=new BIT_VECTOR("1001");

BITVEC1 chỉ đến đối tượng được ràng buộc 4 bits .Bits có thể được xem như BITVEC1(0),là 1,là 0 và cứ như thế .Đây là thí dụ khác nơi mà những giá trị được gán cho đối tượng tương tự sử dụng tên kết hợp
MOD2PTR:=new MODUE'(CRITICAL_DLY=>10 ns,
NO_INPUTS=>2,NO_OUTPUTS=>SIZE=>100);

Kiểu access có lợi trong kiểu mẫu hành vi mức độ cao ,đặc biệt trong cấu trúc thường gặp như RAMS và FIFOs, nơi pointers có thể sử dụng đến những đối tượng access tức thời trong phối hợp .

3.3.5 INCOMPLETE TYPES

Có thể có một kiểu access chỉ đến 1 đối tượng mà đối tượng có những phần tử cũng là kiểu access .Điều này có thể được chỉ đạo phụ thuộc lẫn nhau hoặc trở lại kiểu access.Vì thế 1 kiểu phải được khai báo trước khi sử dụng ,một khai báo kiểu không hoàn toàn có thể được sử dụng để cứu cho vấn đề này .Một kiểu khai báo không hoàn toàn có dạng như sau :

Type type_nam ;

chỉ một kiểu không đầy đủ được ,type_nam có thể được sử dụng trong một kiểu phụ thuộc lẫn nhau hoặc 1 kiểu access ngược (recursive access type), dù sao một khai báo kiểu đầy đủ tương ứng phải theo sau đó .Một thí dụ của một kiểu access phụ thuộc access lẫn nhau là:

type COMP;-- record chứa đựng bao gồm tên và danh sách mạng lưới mà nó liên lạc tới .
type NET; -- record chứa đựng bao gồm tên và danh sách component mà nó liên lạc tới .
type COMP_PTR is access COMP;
type NET_PTR is access NET;
constant MODMAX : INTEGER:=100;
constant NETMAX :INTEGER:=2500;
type COMP_LIST is array (1 to MODMAX) of COMP_PTR;
type NET_LIST is array (1 to NETMAX) of NET_PTR;

```
type COMPLIST_PTR is acccess COMP_LIST;
type NETLIST_PTR is access NET_LIST;
---Kiểu khai báo  đầy đủ cho COMP và NET như sau :
type COMP is
          record
                    COMP_NAME:STRING(1 to 10);
                    NET:NETLIST_PTR;
          end record;
type NET is
          record
                    NET_NAME (1 to 10);
                    COMPONENTS:COMPLIST_PTR;
          end record;
```

Ở đây ,COMPvà NET có những phần tử là những đối tượng access của kiểu NET và COMP ,tương ứng .Một thí dụ của kiểu access recursive là:

```
type DFG ;        --- array data flow graph node.
Type OP_TYPE is (ADD,SUB,MUL,SHIFT,ROTATE);
type PTR is access DFG;
type DFG is
          record
                    OP_CODE :OP_TYPE;
                    SUCC :PTR                --successor node list.
                    FRED:PTR                 --predecessor node list.
          End record;
```

PTR là ot kiểu access của DFG.nó cũng là kiểu của 1 phần tử trong DFG.
Sau đây là những thí dụ khác :

```
type MEM.RANGE is rang 0 to 16383;
type HOLE                         --represent available memory.
 Type HOLE_PTR is access HOLE;
type HOLE is
          record
                    START_LOC,END_LOC:MEM_RANGE;
                    PREV_HOLE,NEXT_HOLE:HOLE_PTR;
          end record;
```

3.3.6 FILE TYPES

Đối tượng của những kiểu file đại diện cho những tập tin trong môi trường chính .Chúng cung cấp một cơ cấu bởi 1 loại truyền đạt thông tin thiết kế VHDL với môi trường chính .Cú pháp của khai báo kiểu tập tin là:
type file_type_nam is file of type_name;
Type_nam là một kiểu của những giá trị chứa đựng trong một file .

Sau đây là 2 thí dụ :

```
type VECTOR is file of BIT_VECTOR;
type NAMES is file of STRING;
```

Một tập tin của kiểu VECTORS có những giá trị   liên tiếp kiểu BIT_VECTOR ;Một file kiểu NAMES có liên tục những chuỗi như giá trị trong nó .
Một tập tin có thể mở,đóng ,đọc ,viết,hoặc kiểm tra điều kiện end-of-file bởi việc sử dụng những thủ tục và hàm đặc biệt ,khai báo ngầm cho mỗi tập tin .

Đó là :

procedure FILE_OPEN (file F: fiel_type-name;
EXTERNAL_NAME :in STRING;
OPEN_KIND:in FILE_OPEN_KIND:=READ_MODE);
--mở một tập tin F chỉ đến tập tin vật lý trong chuỗi EXTERNAL_NAME ,với specified mode.
--Kiểu FILE_OPEN_KIND, có những giá trị : READ_MODE(mặc định),WRITE_MODE,và APPEND_MODE
procedure FILE_OPEN(STATUS :out FILE_OPEN_STATUS;
file F :file-type-name;
EXTERNAL_NAME: in STRING;
OPEN_KIND: in FILE_OPEN_KIND:=READ_MODE);
--Tương tự như thủ tục thứ nhất ,cho rằng thủ tục này trả về trang thái mở tập tin

.K FILE-OPEN-STATUS có những giá trị như sau :
OPEN_OK :tập tin mở thành công .
STATUS_ERROR:tập tin đã mở rồi
NAME_ERROR : tập tin không tìm thấy hoặc không cho phép
MODE_ERROR :không thể mở tập tin với specified access mode.

Procedure FILE_CLOSE(file F:file-type-name);
--closes the specified file.
Procedure READ(file F:file-type-name,VALUE :out type-name);
--Lấy những giá trị sau trong VALUE từ tập tin F.
Procedure WRITE(file F:file-type-name,VALUE :in type-name);
--nối giá trị trong VALUE vào file F.
Function ENDFILE(file F:file-type-name) return BOOLEAN;
--trả về falses nếu đọc trên tập tin F sẽ thành công trong việc lấy giá trị khác ,ngược lại trả về true.
Nếu type-nam là một kiểu dãy không ràng buộc ,một thủ tục READ khác được khai báo ngầm ,có dạng như sau :
procedure READ(file F:file-type-name;VALUE :out type-name;
LENGTH:out NATURAL);
--LENGTH trả về số phần tử của dãy đã được đọc
Những giá trị nằm trong một tập tin có thể chỉ bị ảnh hưởng Một tập tin có thể ngầm mở bởi thông tin mở file trong hkai báo file .Những thí dụ sau thể hiện điều đó

```
file DUMP:NAMES open APPEND_MODE is “top.dump”;
--Một kiểu gọi ẩn đến FILE_OPEN được tạo ra tại thời điểm sinh ra .
--FILE_OPEN(DUMP,”top.dump”,READ_MODE);

file PATTERNS:VECTOR is”uart.pat”
--Một kiểu gọi ẩn đến FILE_OPEN được tạo ra tại thời điểm sinh ra .
--FILE_OPEN(PATTERNS,”uart.pat”,READ_MODE);

file TMP: VECTORS;
```

Vì không có thông tin mở tập tin được cung cấp ,một lần gọi rõ ràng đến FILE_OPEN cần được thực hiện trước khi tập tin TMP có thể bị gia tăng (accessed)
Đây là một thí dụ đầy đủ của một test bench,đọc những vector từ file”fadd.vec”,đặt những vector vào test component , một one-bit full adder,và viết kết quả vào một tập tin khác “fadd.out”.

```
entity FA_TEST is end;
architecture IO_EXAMPLE of FA_TEST is
    component FULL_ADD
        port(CIN, A, B:in BIT;COUTN,SUM:out BIT);
    end component;
subtype STRING3 is BIT_VECTOR(0 to 2);
```

```
subtype STRING2 is BIT_VECTOR(0 to 1);
type  IN_TYPE is file of STRING3;
type  OUT_TYPE is file of STRING2;
file VEC_FILE :IN_TYPE
open READ_MODE is “usr/home/jb/vhdl_ex/fadd.vec”;
fiel RESULT_FILE:OUT_TYPE
opeN WRITE_MODE is “usr/home/jb/vhdl_ex/fadd.out”;
signal S:STRING3;
signal Q:STRING2;
bin
FA:FULL_ADD port map (S(0),S(2),Q(0),Q(1));
process
        constant PROPAGATION_DELAY:TIME:=25ns;
variable IN_STR:STRING3;
variable OUT_STR:STRING2;
begin
        while not ENDFILE(VEC_FILE) loop
READ(VEC_FILE,IN_STR);
S<=IN_STR;
wait for PROPAGATION_DELAY;
OUT_STR:=Q;
WRITE(RESULT_FILE,OUT_STR);
end loop;
report “completed processing all vectors”;
wait;               --stop trường hợp
end process;
end IO_EXAMPLE;
```

Hai tập tin VEC_FILE và RESULT_FILE đã khai báo .VEC_FILE là một tập tin đầu vào và chứa đựng chuỗi 3 bits, và RESULT_FILE là một tập tin xuất là chuỗi 2 bit .Vectors đầu vào được đọc một lần tại một thời điểm cho đến cuối file .Mỗi vector được tham gia va rồi quá trình chờ mạch cộng (full-addder circuit) được ổn định trước khi đơn giản giá trị xuất mạch cộng được ghi vào tập tin xuất .Phát biểu report ,khi thực thi ,in đơn giản thông báo report ,phát biểu wait là nguyên nhân làm qúa trình trì hoãn vô hạn định.
Một kiểu tập tin ,TEXT,được xác định trước trong ngôn ngữ .kiểu tập tin này đại diện phù hợp với những chuỗi text có độ dài thay đổi(variable length).Một kiểu access ,LINE ,cũng chỉ định đến một chuỗi tương tự .Điều khiển đọc và viết những lines cũng được cung cấp .Việc xác định cho tất cả các kiểu và những toán tử xuất hiện trong gói được xác định trước ,TEXTIO.

3.4 <u>OPERATORS</u>

Những toán tử được xác định trước trong ngôn ngữ được phân chia thành 6 loại :

1. Toán tử luận lý (logical operators)
2. Toán tử quan hệ (relation operators)
3. Toán tử xoay (Shift operators)
4. Toán tử cộng (Adding operators)
5. Toán tử nhân (Multiplying operators)
6. Toán tử hỗn hợp (miscellaneuos operators)

Những toán tử có quyền ưu tiên tăng từ loại 1 cho đến 6.Toán tử cùng loại có cùng quyền ưu tiên như nhau ,tính từ trái sang phải .ngoặc đơn có thể được sử dụng để phân định tính từ trái sang phải.

3.4.1 LOGICAL OPERATORS :

có bảy toán tử luận lý là :

and or nand nor xor xnor not

Những toán tử được xác định trước bởi kiểu BIT và BOOLEAN.Chúng cũng có thể được xác định bởi một dãy thứ nguyên của BIT và BOOLEAN.Trong quá trình định lượng của toán tử luận lý .riêng giá trị BIT '0' và '1' được coi như là giá trị FALSE và TRUE của kiểu BOOLEAN.Kết quả của một toán tử luận lý có kiểu tương tự như toán hạng của nó

Toán tử NAND và NOR không kết hợp với nhau ,do đó cú pháp của một biểu thức toán tử liên tục nand và nor là trái luật .Thí dụ biểu thức sau là trái luật

A and B nand C.

Dấu ngoặc đơn có thể sử dụng để tránh trường hợp này .

3..4.2 RELATIONAL OPERATORS (toán tử quan hệ) :

Đó là :

= /= < <= >=

Kiểu kêt quả của tất cả các toán tử quan hệ luôn luôn được xáx định trước là kiểu BOOLEAN.Toán tử bằng(=) và không bằng (/=) luôn được xác định trước trong bất kỳ kiểu nào trừ kiểu FILE .Còn lại 4 toán tử được xác định trước trong kiểu vô hướng bất kỳ (kiểu integer hoặc kiểu liệt kê) hoặc kiểu dãy rời rạc (dãy này có các phần tử trong dãy tùy thuộc vào kiểu rời rạc ,riêng biệt).Khi toán hạng là những kiểu dãy rời rạc .Phép so sánh đóng vai một phần tử tại một thời điểm từ trái sang phải ,thí dụ :

BIT_VECTOR('0','1','1')<BIT_VECTOR('1','0','1')

là thuê ,vì thế phần tử đầu tiên của toàn dãy đầu tiên kém hơn phần tử đầu tiên của toàn dãy thứ hai .Thông thường nếu :

type MVL is('U','0','1','Z');

thì MVL'('U')<MVL'('U')

là đúng vì 'U' xuất hiện bên trái 'Z' .Lưu ý việc cần thiết để định lượng toàn bộ và thực tế sử dụng trong những thí dụ trên vì những literals đã quá mức và kiểu của chúng là không xác định trong việc sử dụng .(Biểu thức định lượng được mô tả trong chương 10)

Sau đây là thí dụ mà toán hạng luôn luôn có chiều dài khác nhau :

"VHDL"<"VHDL92"

là đúng .Sự so sánh một lần nữa đóng vai trò tại một thời điểm từ trái sang phải. Tuy nhiên, phần tử tương ứng không được tạo ra trong toán hạng . Nó coi như là NULL,và NULL luôn luôn được xem như kém hơn bất kỳ ký tự nào .trong thí dụ này, character '9' trong toán hạng thứ hai không tương thích với phần tử trong toán hạng thứ nhất .

3.4.3 SHIFT OPERATORS :

Đó là :

SLL SRL SLA SRA ROL ROR

Mỗi một toán tử giữ một dãy BIT hoặc BOOLEAN như một toán hạng trái và giá trị INTEGER như toán hạng phải ,đóng vai trò giải thích toán tử . Nếu giá trị integer là số âm , hành vi ngược nhau sẽ xảy ra, đó là xoay trái hoặc quay trở thành xoay phải hoặc quay .

toán tử SLL (xoay trái luận lý) và toán tử SRL (xoay phải luận lý) điền vào những bit huỷ bỏ với left-operand-type'LEFT.Toán tử SLA (xoay trái số học) điền vào những bit bị huỷ bỏ với bit cực phải của toán hạng trái .trong khi toán tử SRA (xoay phải số học) điền vào những phần tử bị huỷ bỏ với bit cực trái của toán hạng trái .Toán tử rotate là nguyên nhân những bit bị điền vào với những bit chiếm giữ trong 1 vòng cấu thành .Sau đây là những thí dụ :

--giả sử tất cả những toán hạng trái đều là BIT_VECTOR

"1001010" sll 2 is "0101000" –filled with BIT'LEFT ,which is '0'

"1001010" srl 3 is "0001001"

"1001010"la 2 is "0101000"

. . . .

3.4.4ADDING OPERATORS :

Gồm

+ - &

Toán hạng cho toán tử + và toán tử - đều phải là kiểu số ,và kết quả cũng là kiểu số toán tử + và - cũng được sử dụng như toán tử unary,trong trường hợp toán hạng và kiểu kết quả như nhau .Toán hạng cho toán tử & có thể là kiểu dãy thứ nguyên hoặc kiểu một phần tử .Kết quả luôn luôn là một kiểu dãy .Thí dụ :
'0'&'1' kết quả là kiểu dãy character '01'.

3.4.5 MULIPLYING OPERATORS:

Đó là :

* / MOD REM

Toán tử nhân và chia phải được xác định trước cho cả hai toán hạng phải là cùng kiểu số nguyên hoặc là kiểu floating point .Kết quả cũng phải cùng kiểu .Toán tử nhân cũng được xác định bởi trường hợp khi một trong những toán hạng là kiểu vật lý và toán hạng thứ hai là kiểu integer hoặc real .Kết quả trả về là kiểu vật lý.

Đối với toán tử chia ,chia một giá trị vật lý bởi một giá trị integer hoặc real thì được cho phép.và kết quả trả về là kiểu vật lý .Phép chia của một giá trị kiểu vật lý bởi một đối tượng khác cùng kiểu vật lý và phần còn lại của nó ,một giá trị nguyên coi như một kết quả .

toán tử REM và MOD tác dụng cho toán hạng của kiểu integer và kết quả có cùng một kiểu .Kết quả của REM có biểu hiện của toán hạng thứ nhất và nó được xác định như sau:

A rem B =A -(A / B) * B

Kết quả của toán tử Mod là biểu hiện toán hạng thứ hai .Và nó được xác định như sau :

A mod B =A - B * N –cho một vài số nguyên N

Sau đây là những thí dụ sử dụng toán tử mod và rem :

7 mod 4 --bằng 3
(-7) rem 4 --bằng -3
7 mod (-4) --bằng -1
(-7) rem (-4) --bằng -3

3.4.6 MISCELLANEUOS OPERATORS:

Gồm :
abs **

Toán tử abs (absolute) được xác định cho một kiểu số bất kỳ

Toán tử ** (exponentiatation) xác định cho toán hạng trái là kiểu integer hoặc floating point ,và toán hạng phải phải là kiểu integer.

Toán tử không luận lý có mức độ ưu tiên tương đương như 2 toán hạng trên

CHƯƠNG 4 : BEHAVIORAL MODELING

Chương này đưa ra kiểu hành vi mẫu ,Trong kiểu mẫu này ,hành vi của thực thể được biểu diễn bởi việc thực thi liên tục ,mã thủ tục có cú pháp đơn giản và ngữ nghĩa như ngôn ngữ lập trình cấp cao như C hoặc PASCAL.Phát biểu quá trình có một cơ cấu cơ bản sử dụng hành vi mẫu của thực thể .Chương này mô tả phát biểu chomột qúa trình và thể loại khác nhau của những phát biểu liên tục có thể được sử dụng trong mẫu phát biểu một qúa trình như là hành vi .
Bất kể mẫu sử dụng nào ,mỗi một thực thể miêu tả sử dụng khai báo thực thể và ít nhất 1 architecture body.Hai đoạn đầu tiên sẽ mô tả những chi tiết này .

4.1 ENTITY DECLARATION

Một khai báo thực thể mô tả giao diện bên gnoài của thực thể .nó ghi rõ tên của thực thể ,tên cổng giao tiếp .kiểu mode,và kiểu của các cổng .Cú pháp của một khai báo thực thể là :

```
entity entity-name is
        [generic (list-of-generic-and-their-types);]
        [port(list-of--interface-port-name-and-their-types);]
        [entity-item-declaration]
[begin
        entity-statement]
end [entity][entity-name];
```

entity-name là tên của thực thể ,và cổng giao diện là những tín hiệu đi qua thực thể đưa thông tin đi và đến ra môi trường ngoài .mỗi cổng giao diện có thể có một trong những kiểu sau

1. in :giá trị cổng vào chỉ có thể đọc trong mẫu thực thể
2. out :giá trị cổng ra chỉ có thể được cập nhật mà không thể đọc
3. inout giá trị của cổng điều khiển có thể đọc và cập nhật .
4. buffer: Giá trị của cổng buffer có thể được đọc và cập nhật.Tuy nhiên ,nó khác với inout ở chỗ nó không thể có hơn 1 nguồn ,và chỉ có 1 loại tín hiệu được nối với nó (có thể là một cổng buffer hoặc 1 tín hiệu với nhiều nhất là một nguồn)
5. linkage :Giá trị của cổng liên kết có thể được đọc và cập nhật .Nó chỉ có thể được làm bởi một cổng khác của kiểu liên kết . Cách thông thường của cổng liên kết thì không được sáng sủa và vì vậy không gợi nhớ .Trong quá khứ nó được sử dụng để giao tiếp với ngôn ngữ lạ và ngôn ngữ giả .

Khai báo đặt trong entity-item-declaration là chung cho tất cả đơn vị thiết kế kết hợp với khai báo thực thể này (chúng cũng có thể là architecture bodies và khai báo config).Mạch lật and,or được thể hiện trong hình 4.1,nó tương ứng với khai báo thực thể được chỉ ra dưới đây .

```
entity AOI  is
        port(A,B,C,D:in Bit ;Z:out BIT);
end AOI;
```

Khai báo entify chỉ ra tên của thực thể là AOI và nó có 4 tín hiệu vào thuộc kiểu BIT,một tín hiệu ra hiểu BIT.Chú ý rằng nó không chỉ ra bản chất hoặc chức năng của thực thể .

4.2 ARCHITECTURE BODY:

Một architecture body mô tả bên trong của một thực thể .Nó mô tả chức năng và cấu trúc của thực thể .Cú pháp của một architecture body là :

```
architecture architecture-name of entity-name is
[architecture-item-declaration]
begin
concurrent-statements;these are ->
        process-statement
        block-statement
```

```
            concurrent-procedure-call- statement
            concurrent-assertion- statement
            concurrent-signal-assignment- statement
            component-instantiation-statement
            generate-statement
    end [architecture][architecture-name];
```

Những phát biểu đồng thời mô tả cấu hình bên trong của thực thể .Tất cả những phát biểu đồng thời thực thi song song với nhau ,những thứ tự nguyên bản hiện diện bên trong architecture body thì không tác động tác động lên hành vi bao hàm .Cấu hình bên trong của thực thể có thể được thể hiện trong quan hệ của cấu trúc ,dòng dữ liệu và hành vi liên tục .Chúng được mô tả bằng cách sử dụng những phát biểu đồng thời. Ví dụ,Khai báo thành phần để thể hiện cấu trúc ,phát biểu gán tín hiệu đồng thời được sử dụng để thể hiện dòng dữ liệu và phát biểu quá trình để thể hiện hành vi liên tục .Mỗi một phát biểu đồng thời là một phần tử khác nhau tác động song song ,tương tự như những cổng riên biệt của một thiết kế tác động song song .

Những mục khai báo items là sẵn sàng cho việc sử dụng trong architecture body,tên của những item đã khai báo trong khai báo thực thể ,bao gồm những cổng và những generics, sẵn sàng cho việc sử dụng trong architecture body bởi vì sự kết hợp của tên thực thể với architecture body bởi phát biểu sau :
architecture architecture-nam of entity-name is ...

Một thực thể có thể có rất nhiều views bên trong ,mỗi một view được mô tả architecture body khác nhau .Nói chung ,một thực thể tương ứng với 1 khai báo thực thể (cung cấp view bên ngoài) và một hoặc nhiều architecture bodies(cung cấp view bên trong).Sau đây là 2 thí dụ của architecture body cho cùng 1 thực thể AOI:

```
    architecture AOI_CONCURRENT of AOI is
            begin
            Z<=not((A and B) or (C and Dữ liệu _);
    end AOI_CONCURRENT;
    architecture AOI_SEQUENTIAL of AOI is
            begin
    process(A,B,C,D)
            variable TEMP1,TEMP2:BIT;
            begin
                    TEMP1:=A and B;           --statement 1
                    TEMP2:=C and Dữ liệu  ; --statement2
                    TEMP1:=TEMP1 or TEMP2--statement3
                    Z<=not TEMP
            end process
    end AOI-SEQUENTIAL;
```

Architecture body đầu tiên ,AOI_CONCURRENT,mô tả thực thể AOI sử dụng kiểu dataflow ,Architecture body thứ hai ,AOI_SEQUENTIAL,sử dụng kiểu hành vi .trong nhương này chúng ta quan tâm đến việc mô tả một thực thể sử dụng kiểu hành vi.Phát biểu quá trình là những phát biểu đồng thời ,là cơ chế căn bản được sử dụng để mô tả chức năng của một thực thể trong kiểu này.

4.3 PROCESS STATEMENT:

Những phát biểu process chứa đựng những phát biểu liên tục mô tả chức năng của một phần thực thể trong những thành phần liên tục Cú pháp của một phát biểu process là :

```
[procee-label:]process[(sensitivity-list)][is]
    [process-item-declarations]
begin
    sequential-statements; these are->
            variable-assignment-statement
            wait-statement
```

```
                if-statement
                case- statement
                loop- statement
                null- statement
                exit- statement
                next- statement
                assertion- statement
                report- statement
                procedure-call- statement
                return- statement
end process[process-label]
```

Architecture body ,AOI_SEQUENTIAL,hiên diện sớm hơn ,chứa một phát biểu quá trình .Phát biểu quá trình này có 4 tín hiệu trong danh sách độ nhạy của nó và một khai báo biến .nếu 1 sự kiện xảy ra trên một tín hiệu nào đó thì quá trình được thực thi.Điều này được hoàn thành bởi phát biểu thực thi 1 đầu tiên ,kế đó là phát biểu 2 ,sau nữa là phát biểu 3,rồi đến phát biểu 4 .Sau đó ,quá trình trì hoãn vô hạn định và chờ cho đến một sự kiện khác xảy ra trên trên một tín hiệu trong danh sách độ nhạy .

4.4 VARIABLE ASSIGNMENT STATEMENT:

Biến có thể được khai báo vàsử dụng bên trong một phát biểu quá trình .một biến được gán một giá trị sử dụng phát biểu gán biến ,mà phát biểu này có hình thức như sau :

```
variable-object:=expression;
```

Biểu thức được xác định giá trị khi phát biểu được thực thi và giá trị được tính toán được gán cho biến 1 cách tức thời.

Biến được tạo tại thời điểm sản sinh và duy trì giá trị của nó trong suốt thời gian chạy chương trình (như trong C).Điều này bởi vì quá trình không bao giờ được thoát trong mỗi trạng thái active,nghĩa là được thực thi,hoặc trong 1 trạng thái trì hoãn ,nghĩa là ,chờ cho đến khi một sự kiện chắc chắn xảy ra.Một quá trình bắt đầu bước vào tại điểm khởi đầu của simulation.tại thời điểm này nó được thực thi cho đến khi bị trì hoãn bởi 1 phát biểu wait hoặc 1 sensitivity list .

Xem thí dụ về phát biểu quá trình sau :

```
process(A)
        variable EVENT_ON_A:INTEGER:= -1;
        begin
                EVENT_ON_A:=EVENT_ON_A+1;
        end process;
```

Tại điểm đầu của simulation,Quá trình được thực thi một lần .Biến EVENT_ON_A được gán giá trị -1 sau đó tăng lên 1 .Sau đó ,thời điểm bất kỳ xảy ra sự kiện trên tín hiệu A,quá trình có hiệu lực và phát biểu gán biến đơn được thực thi.Nó làm cho biến EVENT_ON_A tăng lên 1 .Tại thời điểm kết thúc của simulation,biến EVENT_ON_A chứa tổng số sự kiện xảy ra trên tín hiệu A

Sau đây là một thí dụ khác của phát biểu quá trình :

```
        signal A,Z:INTEGER;
        . . .
        PZ:process(A);  --PZ là nhãn của quá trình
        variable V1,V2:INTEGER;
                begin
                        V1:=A-V2;--statement 1
                        Z<= -V1;----statement 2
                        V2:= Z+V1=2; --statement 3
       end process PZ;
```

Nếu một sự kiện xảy ra trên tín hiệu A tại thời điểm T1 và biến V2 được gán giá trị là 10,trong phát biểu thứ 3,sau đó một sự kiện xảy ra trên tín hiệu A tại thời điểm T2, giá trị của V2 được sử dụng trong phát biểu 1 sẽ cũng là 10

Một biến cũng có thể được khai báo bên ngoài 1 quá trình hoặc 1 chương trình con .Một biến có thể được đọc và cập nhật bởi 1 hoặc có thể nhiều quá trình ,những biến này được gọi là *shared variable*

4.5 SIGNAL ASSIGNMENT STATEMENT:

Tín hiệu được gán giá trị sử dụng một phát biểu gán tín hiệu ,hình thức đơn giản của phát biểu gán tín hiệu là :
signal-object<=expression [after delay-value]

Phát biểu gán tín hiệu có thể xuất hiện bên trong hoặc bên ngoài một quá trình .Nếu nó xảy ra bên ngoài của một quá trình ,nó được xem là một phát biểu gán tín hiệu đồng thời .Khi phát biểu gán tín hiệu xuất hiện bên trong quá trình ,nó được xem như là 1 phát biểu gán tín hiệu có thứ tự và nó được thực thi tuần tự theo thứ tự của những phát biểu tuần tự khác xuất hiện bên trong quá trình
Khi 1 phát biểu gán tín hiệu được thực thi,giá trị của biểu thức được tính toán ,và giá trị này được và giá trị này được chuẩn bị để gán cho tín hiệu sau khi delay.Lưu ý rằng biểu thức được định lượng tại thời điểm phát biểu được thực thi và không thực thi sau delay.Nếu không sau khi mệnh đề được làm rõ ,delay được xem như delay delta mặc định.
Một vài thí dụ của phát biểu gán tín hiệu là :
COUNTER<=COUNTER+”0010”---gán sau delta delay
PAR<=PAR xor DIN after 12 ns
Z=(A0 and A1) or (B0 and B1) or (C0 and C1) after 6 ns;

DELTA DELAY :

Delta delay là một delay rất nhỏ. Nó không tương xứng với bất kỳ một delay thực nào và thời gian thực tế thì không gia tăng. Delay này làm mẫu phần cứng tại nơi thời gian tối thiểu cần cho một thay đổi xảy ra, Delta delay cho phép thứ tự sự kiện xảy ra tại cùng một thời điểm .Mỗi đơn vị thời gian mô phỏng có thể được xem là bao gồm một số vô hạn của delta delay.Do đó sự kiện luôn luôn xảy ra tại thời gian thực công với 1 b61I số tích phân của delta delay,Thí dụ ,sự kiện có thể xảy ra tại 15 ns,15ns+1Δ,15ns+2Δ,15ns+3Δ,22 ns,22ns+Δ,27 ns,27ns+Δ.
Giả sử có architecture body được mô tả trong hình 4.2. chúng ta hãy giả sử rằng một sự kiện xảy ra trên tín hiệu nhập D (có một thay đổi giá trị trên tín hiệu D) tại thời điểm mô phỏng T. Phát biểu 1 được thực thi đầu tiên, và TEMP1 được gán giá trị lập tức bởi vì nó là biến. Phát biểu 2 được thực thi kế tiếp, và TEMP2 được gán giá trị lập tức. Phát biểu 3 được thực thi kế tiếp, nó dùng giá trị của TEMP1 và TEMP2 đã được tín trong phát biểu 1 và 2 để xác định giá trị mới cho TEMP1. Cuối cùng, phát biểu 4 được thực thi làm cho tín hiệu Z lấy giá trị của vế phải biểu thức sau khi delta delay, nghĩa là, tín hiệu chỉ có giá trị tại thời điểm T +Δ.
Xem xét quá trình PZ được mô tả trong đoạn trước .Nếu 1 sự kiện xảy ra trong tín hiệu A tại thời điểm T,thực thi phát biểu 1,nguyên nhân V1,được 1 giá trị ,tín hiệu được dự định đạt giá trị tại thời điểm T+Δ,và cuối cùng tín hiệu thứ 3 được thực thi với giá trị cũ của tín hiệu Z đã được sử dụng ,đó là giá trị tại thời điểm T,không có giá trị nào được dự định gán cho phát biểu thứ hai .nguyên nhân do tại thời điểm đó cho đến thời điểm T không xảy ra lệch pha với thời gian T+Δ.Sau đó khi đến thời gian T+Δ ,tín hiệu Z sẽ nhận một giá trị mới .Điều này rất quan trọng trong việc phân biệt giữa phép gán biến và 1 phát biểu gán biến .Những phép gán biến là nguyên nhân thay đổi tức thời những giá trị của chúng .trong khi phép gán tín hiệu luôn luôn là nguyên nhân tín hiệu đặt những giá trị của chúng tại những thời điểm sau này .
Xa hơn nữa chúng ta sẽ thấy 2 thí dụ của những phát biểu tuần tự ,phát biểu gán biến và phát biểu gán tín hiệu.Những loại khác của phát biểu tuần tự sẽ được mô tả sau

4.6 WAIT STATEMENT :

Như chúng ta đã thấy ,1 quá trình có thể trì hoãn bởi 1 list sensitiviy.Đó là .Khi quá trình có một sensitivity list .nó luôn luôn trì hoãn sau khi thực thi phát biểu tuần tự cuối cùng trong quá trình .Phát biểu wait cung cấp một cách luân phiên trì hoãn công đoạn thực thi 1 quá trình .Có 3 hình thức cơ bản của phát biểu wait .

wait on sensitivity-list;
wait on until boolean -expression;

```
wait for time-expression;
```

Chúng cũng có thể được phối hợp trong 1 phát biểu wait đơn độc .Thí dụ như :

```
wait on sensitivity-list until boolean-expression for time-expression
```

Những thí dụ của phát biểu wait là :

```
  wait on A,B,Có thể ;
  wait until A=B;
  wait for 10 ns;
 wait on CLOCK for 20 ns ;
 wait until SUM>100 for 50 ms;
```

Sự hiện diện của sensitivity list trong một quá trình mang dáng vẻ của phát biểu "wait on sensitivity-list" như phát biểu cuối cùng trong process .Một phát biểu quá trình định lượng(equivalent process statement) cho phát biểu quá trình trong architecture body AOI_SEQUENTIAL là :

```
process                    --no sensitivity list
        variable TEMP1,TEMP2:BIT;
begin
        TEMP1:=A and B;
        TEMP2:=C and D;
        TEMP1:=TEMP1 or TEMP2;
        Z<=not TEMP1;
        wait on A,B,C,D;        --replaces the sensitivity list.
End process.
```

Vì thế ,1 quá trình với 1 sensitivity list luôn luôn trì hoãn phần cuối của quá trình ,và khi tác động trở lại cho sự kiện ,lại bắt đầu thực thi từ phát biểu đầu tiên trong quá trình .

4.7 IF STATEMENT:

Một phát biểu if lựa chọn những phát biểu tuần tự cho việc thực thi trên giá trị của một kiểu kiện ,điều kiện ở đây có thể là : một biểu thức bất kỳ mà giá trị phải là kiểu luận lý .

Dạng thông tường của phát biểu if là :

```
if boolean-expression then
        sequential-statements
        {elsif boolean-expression then   --mệnh đề elsif.Nếu phát biểu if có 0              sequential          -
statement  }                             hơn một mệnh đề elsif
        {else
                sequential-statement}    -- mệnh đề else
        enf if;
```

Thí dụ :

```
if sum<=100 then –"<=" is less-than-or-equal-to operator.
SUM:=SUM+10;
end if;
```

4.8 CASE STATEMENT :

Dạng của phát biểu case là :

```
case expression is
        when choices=> sequential -statement –branch 1#
        when choices=> sequential -statement –branch 2#
--có thể có nhiều nhánh
        {when others=> sequential-statement}—last branch
end case;
```

Phát biểu case lưa chọn một trong những nhánh cho việc thực thi dựa trên giá trị của biểu thức. Giá trị biểu thức phải thuộc kiểu trừu tượng hoặc kiểu dãy một chiều. Sự chọn lựa có thể được thể hiện như giá trị đơn, vùng giá trị bởi việc sử dụng dấu | hoặc sử dụng mệnh đề khác. Tất cả các giá trị có thể có của biểu thức phải được thể hiện trong phát biểu case đúng 1 lần. Các mệnh đề khác có thể được sử dụng để bao quát tất cả các giá trị, và nếu có, phải là nhánh cuối cùng trong phát biểu case. Một thí dụ cho phát biểu case:

```
type WEEK_DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);
type DOLLARS is rang 0 to 10;
varieble DAY: WEEK_DAY;
variable POCKET_MONEY: DOLLARS;
case DAY is
        when TUE => POCKET_MONEY :=6;            --branch1
        when MON | WED  => POCKET_MONEY :=2;--branch2
        when FRI to SUN => POCKET_MONEY :=7;   --branch3
        when others => POCKET_MONEY :=0;                    --branch4
end case;
```

Nhánh 2 được chọn nếu DAY có giá trị là MON hoặc WED. Nhánh 3bao gồm các giá trị FRI, SAT và SUN. Trong khi nhánh 4 gồm các giá trị còn lại, THU. Phát biểu case cũng là phát biểu tuần tự, tuy nhiên nó cũng có thể được xếp lồng vào nhau.

4.9 NULL STATEMENT

phát biểu ***null***

là một phát biểu tuần tự không gây ra bất kỳhành động nào; tiếp tục thực thi với phát biểu kế tiếp. Một thí dụ cho việc sử dụng phát biểu này là trong phát biểu if hoặc trong phát biểu case.

4.10 LOOP STATEMENT:

Một phát biểu lặp được sử dụng để lặp lại một loạt các câu lệnh tuần tự. Cú pháp của phát biểu lặp là:

```
[loop-label:] iteration-scheme loop
        sequential-statements
end loop [loop-lebel];
```

Có 3 kiểu sơ đồ lặp. Đầu tiên là sơ đồ lặp có dạng:

```
for  identifier in range
```

Một thí dụ cho sơ đồ lặp này là:

```
 FACTORAL:=1;
for NUMBER in 2 to N  loop
      FACTORAL :=FACTORAL*NUMBER;
enf loop;
```

Trong thí dụ này .thân của vòng lặp thực thi N-1 lần ,với danh hiệu lặp,NUMBER,tăng lên 1 sau mỗi vòng lặp ,đối tượng NUMBER được khai báo ẩn trong vòng lặp tùy thuộc vào kiểu integer,nó có giá trị từ 2 đến N ,vì vậy khai báo không rõ ràng cho danh hiệu vòng lặp là điều cần thiết ,danh hiệu vòng lặp cũng không thể được gán cho bất kỳ giá trị nào trong vòng lặp for .Nếu 1 biến khác có cùng tên được tạo bên ngoài vòng lặp for,đó là hai loại biến được giải quyết riêng rẽ và biến sử dụng trong vòng lặp for sẽ chuyển giao cho danh hiệu vòng lặp .

Vùng của vòng lặp FOR cũng có thể là vùng của một kiểu liệt kê

Thí dụ :

```
type HEXA is (‘0’,’1’,’2’,’3’,’A’,’B’,’C ’);
. . . .
for NUM in HEXA’(‘2’) downto HEXA’(‘0’) loop
```

```
        --Num sẽ lấy những  giá trị  trong kiểu HEXA từ 2 cho đến 0
end loop
```

4.11 EXIT STATEMENT:

Phát biểu exit là một phát biểu tuần tự nó chỉ có thể sử dụng bên trong vòng lặp .Là nguyên nhân thực thi việc nhảy đến tận cùng vòng lặp hoặc khỏi vòng lặp khi nhãn được xác định .Cú pháp của một phát biểu exit là :

exit [loop-label][when condition]

Nếu nhãn vòng lặp không rõ ràng thì lặp đến tận cùng .Nếu mệnh đề WHEN được sử dụng mệnh đề rõ ràng sẽ được tạo ra ,nếu điều kiện là đúng ,ngược lại ,việc thực thi sẽ tiếp tục với phát biểu kế tiếp.Một hình thái so le cho vòng lặp 2 được mô tả ở đoạn trước là :

```
        SUM:=1;J:=0;
        L3:loop
        J:=J+21;
        SUM:=SUM*10
if (SUM>100) then
        exit L3;
enf if;
end loop l3;
```

4.12 NEXT STATEMENT:

Phát biểu next cũng là phát biểu liên tục cũng chỉ có thể sử dụng bên trong vòng lặp .Cú pháp tương tự như phát biểu exit :

next [loop-label][when condition];

kết quả của phát biểu next bỏ qua những phát biểu còn lại trong lần lặp hiện tại của vòng lặp rõ ràng ,tiếp tục thực thi với phát biểu đầu tiên trong vòng lặp kế tiếp ,nếu tồn tại một lần ,nếu nhãn vòng lặp không rõ ràng ,xảy ra việc lặp đến tận cùng .Đối lập với phát biểu exit ,nó là nguyên nhân vòng lặp bị giới hạn

Thí dụ :

```
for J in 10 downto 5 loop
if SUM<TOTAL_SUM then
        SUM:=SUM+2;
elsif SUM:=TOTAL_SUM then
        next;
else
        null;
end if;
K:=K+1;
end loop;
```

Khi phát biểu next được thực thi ,thực thi việc nhảy đến phần cuối của vòng lặp (phát biểu cuối cùng K:=K+1,thì không thực thi),giảm giá trị của danh hiệu vòng lặp ,J,và làm lại .

4.13 ASSERTION STATEMENT:

Phát biểu assertio thì hữu ích cho ràng buộc 1 thực thể .Thí dụ bạn có thể muốn kiểm tra :nếu giá trị tín hiệu không nằm trong vùng khai báo hoặc kiểm tra việc cài đặt và định giờ cho tín hiệu đến cổng vào của thực thể .Nếu kiểm tra sai ,thông báo sẽ xuất hiện .Cú pháp của phát biểu assertion là :

```
   assert boolean-expression
        [report string- expression]
        [severity expression ];
```

Nếu giá trị của biểu thức boolean là sai .Thông báo sẽ được chuyển đi ở mức độ đơn giản .Biểu thức trong mệnh đề severity phải là giá trị của kiểu SEVERITY_LEVEL (kiểu liệt kê được xác định trước với những giá trị NOTE,WARNING,ERROR và FAILURE) .Mức độ SEVERITY là kiểu sử dụng bởi mô phỏng hành vi chiếm hữu ban đầu tùy thuộc vào giá trị của nó .Thí dụ Nếu mức SEVERITY là ERROR .Mô phỏng có thể thoái lui khỏi quá trình mô phỏng và cung cấp những thông tin chuẩn đoán xác đáng .Tại mức độ thấp nhất ,mức độ SEVERITY mới được hiện ra

4.14 REPORT STATEMENT :

Một phát biểu report có thể được sử dụng để thể hiện một thông báo .Nó tương tự như phát biểu assertion .Nhưng không kiểm tra assertion .Cú pháp có dạng như sau :

```
report string- expression
[severity expression]
```

Khi biểu thức là mệnh đề severity ,nó gây ra việc in một chuỗi ,và mức severity được báo đến mô phỏng cho hành vi tương ứng .Sau đây là thí dụ :

```
if CLR='Z' then
        report "signal CLR  has a high-impedance value";
        --Mức độ SEVERITY mặc định là NOTE
end if;
if CLK/='0' and CLK/='1' then
        report "CLK is neither a '0' nor a '1'!!!!"
        severity ERROR
end if
```

CHƯƠNG 5 : DATAFLOW MODELING

Chương này thể hiện kỹ thuật cho việc thiết kế 1 dataflow cho một thực thể.Một dataflow thiết kế rõ ràng chức năng của một thực thể mà không rõ về cấu trúc .Chức năng thể hiện trong dòng thông tin trong suốt thực

thể .Nó thể hiện căn bản sử dụng những phát biểu gán tín hiệu đồng thời và những phát biểu khối ..Đây là điều ngược lại với kiểu hành vi của việc mô tả thiết kế trong chương trước ,trong đó chức năng của một thực thể được thể hiện bằng các sử dụng những phát biểu thực thi tuần tự .Chương này cũng mô tả chức năng giải quyết và cách sử dụng của chúng .

5.1 CONCURRENT SIGNAL ASSIGNMENT STATEMENT (PHÁT BIỂU GÁN TÍN HIỆU ĐỒNG THỜI) :

Một trong những kết cấu cơ bản cho việc thiết kế hành vi dòng chảy dữ liệu của một thực thể là sử dụng phát biểu gán tín hiệu đồng thời .Một thí dụ của thiết kế dòng chảy dữ liệu cho 2 cổng vào được thể hiện ở hình 5.1 như sau :

```
entity OR2 is
port(signal A,B:in BIT;signal Z:out BIT);
end OR2;
architecture OR2 of OR2 is
begin
        Z<=A or B after 9 ns;
end OR2;
```

Architecture body chứa một phát biểu gán tín hiệu duy nhất nó được thể hiện trong dòng chảy dữ liệu của cổng OR.Việc giải thích của phát biểu này là bất kỳ lúc nào xảy ra sự kiện (thay đổi giá trị)Trong cả tín hiệu A hoặc tín hiệu B (A và B đều là tín hiệu trong biểu thức cho Z),Biểu thức bên phải được định lượng và giá trị của nó được sắp xếp để xuất hiện trên tín hiệu Z sau khi trễ 9 ns .Tín hiệu trong biểu thức A và B là dạng “sensitivity list “ cho phát biểu gán tín hiệu .

Có hai điểm cần lưu ý trong thí dụ này :

1. Cổng ra và cổng vào điếu là dạng tín hiệu đã được khai báo rõ trong khi khai báo thực thể (đây là giá trị mặc định dù nếu không khai báo).
2. Tên của architecture và tên của thực thể (entity) giống nhau ,điều này không thành vấn đề ,vì những architecture bodies là đơn vị thứ cấp .trong khi khai báo thực thể là đơn vị cơ bản và ngôn ngữ cho phép đơn vị thứ cấ có cùng tên với đơn vị cơ bản .

Một architecture body có thể chứa số lượng bất kỳ của những phát biểu gán tín hiệu đồng thời .Vì chúng là những phát biểu đồng thời ,nên thứ tự của những phát biểu là không quan trọng . Những phát biểu gán tín hiệu đồng thời được thực thi bất cứ khi nào sự kiện xảy ra trong tín hiệu được sử dụng trong biểu thức .Một thí dụ cho thiết kế dòng dữ liệu cho 1-bit-full-adder.

```
entity FULL-ADDER is
        port (A,B,CIN:in BIT;SUM,COUT:out BIT);
end FULL_ADDER;
architecture FULL_ADDER of FULL_ADDER is
bin
        SUM<=A xor B xor CIN after 15 ns ;
COUT<=(A and B) or (B and CIN) or (CIN and A) after 10 ns
end FULL_ADDER;
```

Hai phát biểu gán tín hiệu đã thể hiện dòng dữ liệu trong thực thể FULL_ADDER.Bất cứ khi nào một sự kiệ xảy ra trong tín hiệu A,B hoặc CIN .Biểu thức của cả hai phát biểu sẽ được tính ,và giá trị SUM sẽ được xuất hiện sau 15 ns ,và giá trị COUT được tính sau 10 ns Mệnh đề after thiết kế việc trì hoãn luận lý thể hiện bởi biểu thức .

Ngược lại với phát biểu xuất hiện bêb trong phát biểu quá trình .Phát biểu bên trong quá trình sẽ được thực thi tuần tự .Khi phát biểu bên trong một architecture body đều là những phát biểu đồng thời và thứ tự độc lập .Bản thân của phát biểu quá trình là phát biểu đồng thời ,có nghĩa là nếu có bất kỳ phát biểu gán tín hiệu đồng thời nào và phát biểu quá trình bên trong 1 architecture body ,thứ tự của những phát biểu cũng không thành vấn đề .

5.2 CONCURRENT VERSUS SEQUENTIAL SIGNAL ASSIGNMENT:

Trong chương này chúng ta đã thấy phát biểu gán tín hiệu cũng có thể xuất hiện bên trong thân của một phát biểu process như phát biểu gọi phát biểu gán tín hiệu tuần tự .khi phát biểu gán tín hiệu xuất hiện bên ngoài của một process gọi phát biểu gán tín hiệu tuần tự. phát biểu gán tín hiệu đồng thời là một sự kiện trigger (event-triggered).Đó là nó thực thi bất kỳ lúc nào có một sự kiện trên tín hiệu xuất hiện bên trong biểu thức của nó .Khi phát biểu gán tín hiệu tuần tự không là event-triggeredvà thực thi tuần tự trong quan hệ của những phát biểu tuần tự khác xuất hiện bên trong process .

Bên cạnh sự khác nhau trên phát biểu gán tín hiệu đồng thời là đồng nhất với phát biểu gán tín hiệu tuần tự trong những số hạng của hành vi của chúng

Đối với mỗi phát biểu gán tín hiệu đồng thời có một phát biểu quá trình tương đương về mặt ngữ nghĩa .

Phát biểu gán tín hiệu đồng thời

```
CLEAR<=RESET or PRESET after 15 ns;
--RESET and PRESET are signals
```

tương đương với phát biểu sau :

```
process
    begin
        CLEAR<=RESET or PRESET after 15ns
        wait on RESET,PRESET;
end process;
```

Phát biểu gán tín hiệu đồng nhất xuất hiện bên trong một phát biểu process với 1 phát biểu wait ,chúng củasensitivity list bao gồm những tín hiệu được sử dụng trong biểu thức của phát biểu gán tín hiệu đồng thời

Một phát biểu gán tín hiệu đồng thời có thể tạo như trì hoãn .Như:

CLEAR <=postponee RESET or PRESET after 15 ns ;

Ngữ nghĩa của phát biểu trên thì đồng nhất với ngữ nghĩa của phát biểu process đồng nhất của nó.Nghĩa là một process trì hoãn.

5.3 DELTA DELAY REVISITED :

Trong một phát biểu gán tín hiệu nếu không có trì hoãn hoặc trì hoãn là 0 ns ,Một delta delay được đặt ra .Delta delay là một tổng số thời gian vô cùng nhỏ .Nó không là thời gian thực và không gây ra thời gian mô phỏng thực để thay đổi .Cơ cấu delta delay cung cấp thứ tự sự kiện có thể xảy ra trong cùng một thời gian mô phỏng .

Chương 6 : STRUCTURE MODELING

Chương này mô tả kiểu cấu trúc của mô hình . Thực thể là mô hình , là kiểu tập hợp các thành phần kết nối bởi các tín hiệu , nó như là 1 mạng lưới (netlist). Hành vi của entity không thể hiện rõ từ mô hình của nó . Các phát biểu component instantiation là cơ chế chủ yếu được sử dụng cho việc mô tả 1 mô hình của entity.

6.1 MỘT VÍ DỤ :

Bao gồm mạch được chỉ ra ở hình 6.1 và cấu trúc mô hình VHDL của nó là :

```
entity GATING is
    port (A,CK,MR,DIN :in BIT ; RDY,CTRLA :out BIT);
end GATING;
architecture STRUCTURE_VIEW of GATING is
    component  AND2
        port (X,Y:in BIT; Z : out BIT);
```

```
        end component;
        component  DFF
              port (D,CLOCK:in BIT; Q,QBAR : out BIT);
        end component;
        component  NOR2
              port (DA,DB:in BIT; DZ : out BIT);
        end component;
        signal S1,S2 :BIT;

    begin
        D1: DFF port map (A,CK,S1,S2);
        A1: AND2 port map ( S2,DIN,CTRLA);
        N1: NOR2 port map (S1,MR,RDY);
    End STRUTURE_VIEW
```

Ghi chú :

Q' là QBAR.

> là clock.

Ba component AND2,DFF, và NOR2 được khai báo , các thcomponent này ở trong architecture body qua 3 phát biểu component instantiation và chúng được kết nối qua tín hiệu S1 và S2 . Các phát biểu component instantiation là các phát biểu đồng thời và thứ tự xuất hiện của chúng trong architecture body là không quan trọng.

Tuy nhiên mỗi một component instantiation phải có nhãn component khác nhau , ví dụ A1 là nhãn cho component AND2.

6.2 KHAI BÁO COMPONENT :

Một component instantiation trong mô tả cấu trúc phải khai báo sử dụng 1 component . Thành phần khai báo là khai báo tên và các giao tiếp bên ngoài component. Giao tiếp bên ngoài xác định mô hình và kiểu của các cổng . Cú pháp của 1 form trong khai báo component đơn giản là :

```
component  component_name {is}
     {port (list_of_interface_ports);}
end component { component_name};
```

C*omponent_name* là chỉ dẫn đến tên 1 entity đã tồn tại trong thư viện . Nó có thể hướng đến 1 entity , nếu không thì mô hình không thể mô phỏng được (mô hình có thể chỉ là thiết kế). Thông tin bắt buộc có thể sử dụng 1 configuration (configuration được bàn luận trong chương sau).

List_of_interface_ports xác định tên , mode và type cho từng cổng của component tương tự như trong khai báo entity . tên của các cổng có thể khác với tên của các cổng trong thực thể mà nó hướng đến (tên các cổng khác có thể đặt trong configuration).

Một số ví dụ khai báo component là :

```
component NAND2
    port (A,B :in MVL ; Z :out MVL );
end component;

component MP
    port (CK,RESET,RDN,WRN : in BIT ;
          DATA_BUS : inout INTEGER range 0 to 255;
          ADDR_BUS :in BIT_VECTOR (15 downto 0));
```

```
end component ;

component RX
     port (CK,RESET,ENABLE , DATAIN,RD : in BIT ;
            DATA_OUT : inout INTEGER range 0 to (2**8-1);
            PARITY_ERROR, FRAME_ERROR,
             OVERRUN_ERROR : out BOOLEAN ));
end component ;
```

Các khai báo component xuất hện trong phần khai báo của architecture body , mặt khác chúng có thể xuất hiện traong khai báo package .Khai báo phần tử trong package này có thể nhìn thấy thân 1 architecture nào đó bằng cách sử dụng mệnh đề library và use .

Ví dụ : entity GATING mô tả trong phần trước :

```
Package COMP_LIST is
       component  AND2
              port (X,Y:in BIT; Z : out BIT);
       end component;
       component  DFF
              port (D,CLOCK:in BIT; Q,QBAR : out BIT);
       end component;
       component  NOR2
              port (DA,DB:in BIT; DZ : out BIT);
       end component;
end COMP_LIST;
```

Package này được biên dịch vào thư viện DES_LIB, architecture body có thể viết lại như sau :

```
 library DES_LIB
 use DES_LIB.COMP_LIST.all;
 architecture STRUCTURE_VIEW of GATING is
      signal S1,S2 : BIT;
      -- No need for speccitying component declaration here, since they
      --Are made visible to architecture body
        -- Using library and use clauses.
begin
        --The component instantiation here
end COMP_LIST;
```

Điểm thuận lợi của cách này là package đó có thể tham gia vào các đơn vị thiết kế khác , các khai báo component không cần nằm trong các đơn vị thiết kế .

6.3 ĐỐI TƯỢNG THÀNH PHẦN (component instantiation):

Một phát biểu component instantiation được định nghĩa là phần con của entity khi nó xuất hiện . Nó liên kết các tín hiệu trong entity với các cổng của subcomponent đó . Một phát biểu component instantiation có dạng sau :

component_label : component_name { port map (*association_list*)};

Component_label : có thể là nhận dạng và cũng có thể xem là tên của 1 ví dụ.
Component_name : là tên của component khai báo trước , trong phần khai báo component.

Association_list : liên kết các tín hiệu trong entity , gọi là các số thực (*actual*), với các cổng của component gọi là các số hình thức (*formal*).

Các actual có thể là 1 tín hiệu , hiện thực cho cổng input có thể là 1 biểu thức ,có thể là 1 từ khóa mở đến cổng mà nó không kết nối .

Đây là 2 phương pháp thực hiện sự liên kết của các formal và actual:

1. Liên kết theo vị trí.
2. Liên kết theo tên.
3.

Trong liên kết theo vị trí , association_list có dạng sau :

actual1, actual2,actual3 … , actualn

Mỗi một actual trong component instantiation được ánh xạ theo vị trí với các port trong khai báo component. Port thứ nhất trong khai báo component tương ứng với actual thứ nhất trong component instantiation , tương tự port thứ hai tương ứng với actual thứ hai , cứ thế đến hết danh sách liên kết. Sau đây là 1 ví dụ :

```
--Component declaration:
   component NAND2
        port (A,B:in BIT; Z : out BIT );
   end component;
 --Component instantiation:
   N1: NAND2 port map (S1,S2,S3);
```

N1 là nhãn của component đối với đối tượng hiện hành của component NAND2. Tín hiệu S1 (là 1 actual) liên kết với port A (là 1 formal) của component NAND2, tín hiệu S2 (là 1 actual) liên kết với port B (là 1 formal) của component NAND2, và S3 liên kết với port Z . Tín hiệu S1 và S2 cung cấp 2 giá trị input cho component NAND2 , và tín hiệu S3 nhận giá trị từ port output của component. Thứ tự các số thực ởđây rất quan trọng.

Nếu 1 cổng trong component instantiation không có tín hiệu nào kết nối , có thể sử dụng từ khoá open để biểu hiện cho port không có nối kết . Ví dụ:

```
N3 : NAND2 port map (S1,open,S3);
```

Port input thứ hai của component NAND2 không có tín hiệu nào liên kết . Một port input có thể đưa vào khai báo của chúng giá trị ban đầu . Trong phát biểu component instantiation trước để cho hợp pháp , port B phải có giá trị đầu là 1 biểu thức.

```
component NAND2
     port ( A,B :in BIT :='0' ; Z : out BIT );
    -- both A and B have an initial value of '0' ; however, only
      --the initial value of B is necessary inthis case.
end component;
```

Trong liên kết bằng tên , association_list có dạng sau :

Formal1 => actual1 ,formal2 => actual2 … , formaln => actualn

Ví dụ xem xét component NOR2 trong khai báo entity GATING trong phần trước , có thể viết lại như sau :

```
N1: NOR2 port map ( DB => MR, DZ => RDY , DA => S1);
```

Trong trường hợp này , tín hiệu MR (1 actual) ,đã được khai báo trong danh sách các port của entity, liên kết tới port thứ hai(port DB , là formal) của NOR2 , tín hiệu RDY liên kết với port thứ ba (port DZ) và tín hiệu S1 liên kết với port thứ nhất (port DA) của NOR2. Trong liên kết bằng tên , thứ tự của các liên kết là không quan trọng ,ánh xạ giữa các actual và các formal được xác định một cách rõ ràng . Điểm quan trọng cần chú ý là phạm vi của các formal được giới hạn trong phần ánh xạ của đối tượng cho thành phần đó , ví dụ formal DA, DB, và DZ của component NOR2 chỉ thích hợp trong ánh xạ của đối tượng thành phần NOR2.

Bất cứ kiểu liên kết nào nó cũng chịu ràng buộc bởi ngôn ngữ . Trong kiểu liên kết đầu , các kiểu của formal và actual trong liên kết phải giống nhau. Trong kiểu liên kết thứ hai , các mode của các port phải đáp ứng với qui định ,đó là nếu formal đọc được thì actual cũng phải đọc được , nếu formal viết được thì actual cũng phải viết được . Các tín hiệu khai báo cục bộ xem xét cả hai tính đọc và viết , một tín hiệu có thể liên kết với formal của mode đó . Nếu actual làport của mode in , nó không thể liên kết với formal của mode out hoặc inout , nếu actual làport của mode out , nó không thể liên kết với formal của mode in hoặc inout , nếu actual làport của mode inout , nó có thể liên kết với formal của mode in,out hoặc inout .

Điều quan trọng cần chú ý là 1 actual của mode out hoặc inout cho biết dạng của tín hiệu nguồn , nếu tín hiệu là đa điều khiển thì nó cũng được xác định . Một port đệm không có tín hiệu nguồn , mà nó chỉ liên kết với 1 port đệm khác hoặc tín hiệu chỉ có 1 nguồn .

Một ví dụ khác của component instantiation là :

```
M1: MICRO port map ( UDIN (3 downto 0),WRN,RDN,STATUS(0),
                     STATUS(1),UDOUT( 0 to 7),TXDATA);
```

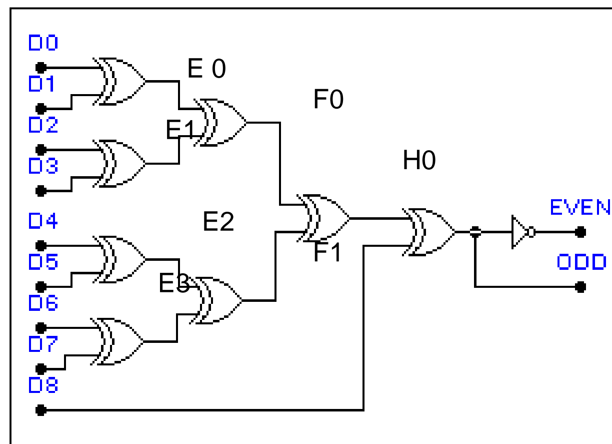
Actual đầu tiên của ánh xạ chỉ đến 1 phần vector tín hiệu UDIN , WRN và RDN là các tín hiệu 1 bit , STATUS(0) và STATUS(1) chỉ đến phần tử thứ 0 và 1 của dãy STATUS , UDOUT(0 to 7) chỉ ra số thành phần của vector UDOUT , và TXDATA chỉ tới toàn bộ 1 vector tín hiệu.

Các ví dụ này chỉ ra rằng ,các tín hiệu sử dụng cho kết nối chung quanh có thể là 1 trong các dạng sau :

+ Slices.
+ Vectors.
+ Array elements.

6.4 CÁC VÍ DỤ KHÁC :

Cấu trúc của mạch parity 9 bit xemhình 6.2 :



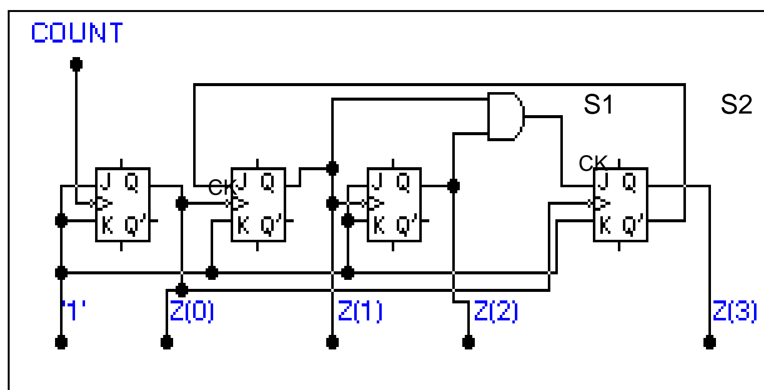
Hình 6.2 A 9 bit parity generator circuit.

```
entity  PARITY_9_BIT is
     port (D :in BIT_VECTOR (8 downto 0);
            EVENT :out BIT ; ODD : buffer BIT );
end  PARITY_9_BIT;

architecture PARITY_STR of PARITY_9_BIT is
         component XOR2
               port ( A,B:in BIT; Z:out BIT);
        end component;
        component INV2
               port ( A:in BIT; Z:out BIT);
        end component;
        signal E0,E1,E2,E3,F0,F1,H0:BIT ;
 begin
        XE0 : XOR2 port map ( D(0), D(1), E0 );
        XE1 : XOR2 port map ( D(2), D(3), E1 );
        XE2 : XOR2 port map ( D(4), D(5), E2 );
        XE3 : XOR2 port map ( D(6), D(7), E3 );
        XF0 : XOR2 port map ( E0, E1, F0 );
        XF1 : XOR2 port map ( E2, E3, F1 );
        XH0 : XOR2 port map ( F0, F1, H0 );
        XODD : XOR2 port map ( H0, D(8), ODD );
        XEVENT: INV2 port map ( ODD,EVENT );
```

Trong ví dụ này ,port ODD là kiểu bộ đệm từ giá trị của port này được đọc cũng như viết vào bên trong architecture . Nếu port này được khai báo là mode inout , thì các tín hiệu định nghĩa bên ngoài của thiết kế PARITY_9_BIT cần có khả năng điều khiển các port này, mặc dù có thể nó không yêu cầu.

Một ví dụ bộ đếm 10 sử dụng J – K flip flop xem hình 6.3 :



Hình 6.3 A decade counter.

```
entity DECADE_CTR is
      port ( COUNT : in BIT ; Z : buffer BIT_VECTOR(0 to 3));
end DECADE_CTR;

architecture NET_LIST of DECADE_CTR is
        component JK_FF
              port ( J,K,CK :in BIT ; Q,Q’ :buffer BIT );
        end component;
```

```
component AND_GATE
      port (A,B :in BIT; C : out BIT );
end component;
signal S1,S2: BIT;
begin
      A1: AND_GATE port map (Z(2),Z(1),S1);
      JK1: JK_FF port map ('1','1', count , Z(0), open);
      JK3: JK_FF port map ('1','1', Z(1) , Z(2), open);
      JK4: JK_FF port map (S1,'1', Z(0) , Z(3), S2);
      JK2: JK_FF port map (S2,'1', Z(0) , Z(1), open);
```

Ví dụ trên sử dụng giá trị constant hoặc biểu thức constant cho actual trong phần ánh xạ.

Các kiểu cấu trúc có thể mô phỏng sau các thực thể mà component mô tả kiểu và vị trí trong thư viện thiết kế . Thực thể o83 mức thấp phải là các kiểu hành vi . Ngữ nghĩa mô phỏng của component instantiation qua ví dụ ta có thể hiểu rõ hơn , Hãy xem component instantiation A1 trong ví dụ trước .

Hành vi tương đương của chúng được mô tả :

```
A1: block                                   -- A block for each instantiation .
      port ( A,B :in BIT ; C : out BIT );   -- Port in component.
                                               -- declaration.
      port map ( C => S1, A =>Z(2) , B => Z(1) ); -- association list.
begin
      AND_GATE : block                      -- the entity block.
         port ( A,B:in BIT ; C :out BIT );   -- port of entity.
         port map (A=>A,B=>B,C=>C);         -- association of component ports
                                                 with entity ports.
            -- declarations that occur in entity declaration and architecture body
             of AND_GATE entity appear here.
         Begin
            -- Behavior in architecture body for AND_GATE entity.
            -- For examble ,
             -- C <= A and B after 10 ns ;
         end block AND_GATE;
   end block A1;
```

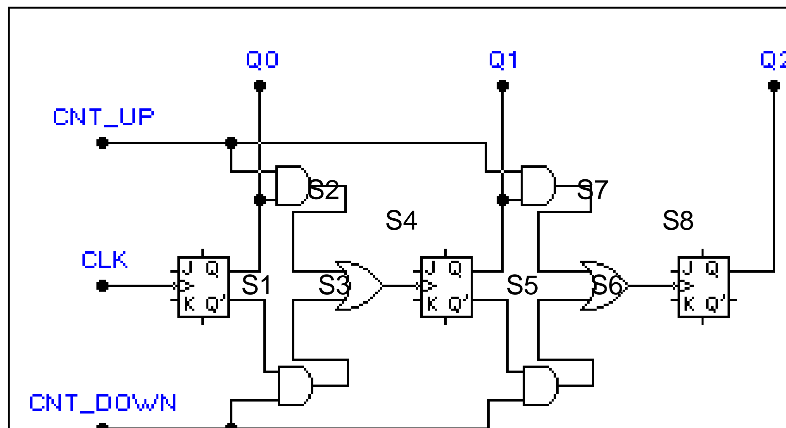
Phát biểu block có thể có 1 danh sách port và 1 port map . Danh sách port được xác định các port, qua đó block trao đổi thông tin với môi trường bên ngoài .

Port map xác định ánh xạ giữa các port và các tín hiệu trong môi trường bên ngoài của block với các port mà chúng kết nối .

Dạng khai báo block được mô tả rõ hơn trong chương 10. Ví dụ sau đây là mạch của bộ đếm lên xuống 3 bit , cấu trúc như sau:

```
entity UP_DOWN is
      port (CLK , CNT_UP,CNT_DOWN : in BIT;
            Q0,Q1,Q2 : buffer BIT );
end UP_DOWN;
architecture COUNTER of UP_DOWN is
      component JK_FF
            port (J,K,CK : in BIT; Q,Q' :out BIT);
      end component;
     component AND2
            port (A,B : in BIT; C :out BIT);
      end component;
```

```
        component OR2
            port (A,B : in BIT; C :out BIT);
        end component;
        signal S1,S2,S3,S4,S5,S6,S7,S8 : BIT;
begin
    JK1: JK_FF por_map (‘1’,’1’,CLK,Q0,S1);
    A1 : AND2 port map (CNT_UP , Q0, S2);
    A2 : AND2 port map ( S1, CNT_DOWN , S3);
    O1 : OR2 port map ( S2, S3, S4);
    JK2: JK_FF por_map (‘1’,’1’,S4,Q1,S5);
    A3 : AND2 port map (Q1 , CNT_UP, S7);
    A4 : AND2 port map ( S5, CNT_DOWN , S6);
    O2 : OR2 port map ( S7, S6, S8);
    JK3: JK_FF por_map (‘1’,’1’,S8,Q2,open);
end COUNTER;
```

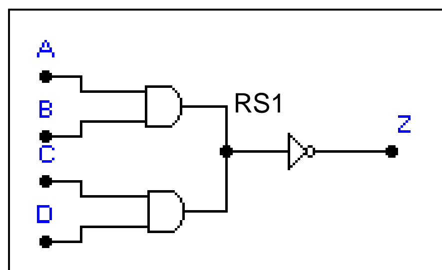


(J,K input of all flip flop connected to ‘1’)

Hình 6.4 A 3 bit up down counter .

6.5 Phân tích các giá trị tín hiệu :

Nếu các output của 2 component điều khiển chung 1 tín hiệu , giá trị của tín hiệu phải được xác định bởi hàm phân tích . tương tự với trường hợp này , tín hiệu được sử dụng gán cho nhiều hơn 1 phát biểu gán tín hiệu đồng thời . Ví dụ giả sử mạch xem trong hình 6.5 , tín hiệu chung RS1 được điều khiển bởi các cổng vào và 2 đường ra ,tín hiệu này được qua cổng đảo tạo nên kết quả Z .



Hình 6.5 Two component driving a common signal.

```
entity  DRIVING_SIGNAL is
      port ( A,B ,C , D : in BIT ; Z : out BIT );
end DRIVING_SIGNAL ;
     -- PULL_UP is the name of a function defined in package
         RF_PACK that has been compiled into the working library .
use  WORK.RF_PACK.PULL_UP;
architecture RESOLVED of DRIVING_SIGNAL is
      signal  RS1 : BIT ;
      component AND2
          port ( IN1,IN2:in BIT ; OUT1 : out BIT );
      end component;
      component INV
          port ( X:in BIT ; Y : out BIT );
      end component;
begin
      A1 : AND2 port map ( A,B,RS1 );
      A2 : AND2 port map ( C,D,RS1 );
      L1 : INV port map ( RS1, Z  );
end  RESOLVED;
```

Điểm then chốt ở đây là khi có sự kiện qua phép gán đến tín hiệu RS1 không làm ảnh hưởng việc sử dụng các phát biểu gán tín hiệu , tín hiệu RS1 được điều khiển bởi 2 cổng output , và do đó phải được xác định bởi hàm phân tích . trong ví dụ trước , hàm phân tích PULL_UP liên quan với tín hiệu RS1 .

Điều đó ý nói rằng các giá trị của các cổng output và giá trị các cổng đã qua hàm phân tích sẽ gán cho tín hiệu RS1 . Mỗi ột cổng out, inout , buffer của component tạo đường điều khiển cho tín hiệu mà nó liên kết.

CHƯƠNG 7 :   ĐẶC ĐIỂM CHUNG & CẤU HÌNH

Chương này giới thiệu các *generics* và các ví dụ để minh họa làm thế nào mà các kiểu thông tin có thể được gửi vào một thực thể (*entity*) dùng các *generics*. Phần sau của chương này thảo luận về sự cần thiết phải định cấu hình (*configuration*) và trình bày hai kỹ thuật xoay chiều (*alternate mechanism*) để định cấu

hình, gọi là đặc tả cấu hình (*configuration specification*) và khai báo cấu hình (*configuration declaration*). Tạo nấc trực tiếp (*direct instantiation*) có thể được sử dụng để không phải viết cấu hình. Đặc điểm này, tùy theo kỹ thuật liên kết trễ (*delay binding mechanism*), cũng được mô tả trong chương này.

7.1. GENERICS

Generics thường hữu dụng để gửi kiểu thông tin từ môi trường vào mô tả thiết kế. Ví dụ như sự tăng giảm độ trễ và kích thước của các cổng giao tiếp. Điều này được thực hiện nhờ các *generic*. Các *generic* của một thực thể (*entity*) được khai báo tùy theo cổng của nó trong khai báo thực thể (*entity declaration*).

Ví dụ: một *generic* N ngõ nhập:

```
entity AND_GATE is
        generic(N:NATURAL);
        port(A:in BIT_VECTOR(1 to N); Z:out BIT);
end AND_GATE;

architecture GENERIC_EX of AND_GATE is
begin
        process (A)
                variable AND_OUT: BIT;
        begin
                AND_OUT := '1';
                for K in 1 to N loop
                        AND_OUT := AND_OUT and A(K);
                        exit when AND_OUT='0';
                end loop;
                Z <= AND_OUT;
        end process;
end GENERIC_EX;
```

Trong ví dụ này, kích thước cổng nhập được thiết kế như là một *generic*. Bằng cách này, chúng ta có thể thiết kế một lớp toàn bộ các cổng với số lượng ngõ nhập biến thiên dùng mô tả đơn. Bây giờ, thực thể (*entity*) AND_GATE có thể được sử dụng với số lượng cổng nhập khác nhau trong các nấc (*instantiations*) khác nhau.

Một *generic* khai báo một đối tượng không đổi (*constant object*) của kiểu (giá trị chỉ đọc) và có thể được dùng trong khai báo thực thể và các phần thân kiến trúc (*architecture body*) tương ứng của nó. Giá trị không đổi này có thể được đặc tả như là một biểu thức tĩnh toàn cục (*globally static expression*) theo một trong các cách sau:

1. Khai báo thực thể (*entity declaration*)
2. Khai báo thành phần (*component declaration*)
3. Tạo nấc thành phần (*component instantiation*)
4. Đặc tả cấu hình (*configuration specification*)
5. Khai báo cấu hình (*configuration declaration*)

Giá trị của một *generic* phải xác định được tại thời điểm xây dựng, nghĩa là, giá trị cho một *generic* phải được chỉ định rõ ràng tại thời điểm sử dụng.

Giá trị của một *generic* có thể được đặc tả trong khai báo thực thể (*entity declaration*) của một thực thể (*entity*), như trong ví dụ trên. Đây là giá trị mặc định của *generic*. Nó có thể được dẫn xuất theo những cách khác.

```
entity NAND_GATE is
        generic(M:INTEGER:=2); -M models the number of inputs.
    port(A:in BIT_VECTOR(M downto 1); Z: out BIT);
end NAND_GATE;
```

Hai cách khác để đặc tả giá trị *generic* là trong khai báo thành phần (*component declaration*) và trong tạo nấc thành phần (*component instantiation*).

Ví dụ:

```
entity ANOTHERGEN_EX is
end;

architecture GEN_IN_COMP of ANOTHER_GEN_EX is
- Khai báo thành phần cho NAND_GATE:
component NAND_GATE
    generic(M:INTEGER);
    port(A:in BIT_VECTOR(M downto 1);Z:out BIT);
end component;
- Khai báo thành phần cho AND_GATE:
component AND_GATE
    generic(N:NATURAL := 5);
    port(A:in BIT_VECTOR(1 to N); Z:out BIT);
end component;
signal S1, S2, S3, S4: BIT;
signal SA: BIT_VECTOR(1 to 5);
signal SB: BIT_VECTOR(2 downto 1);
signal SC: BIT_VECTOR(1 to 10);
signal SD: BIT_VECTOR(5 downto 0);
begin
    - tạo nấc thành phần:
    N1:NAND_GATE generic map (6) port map (SD, S1);
    A1:AND_GATE generic map (N=>10) port map (SC,S3);
    A2:AND_GATE port map (SA, S4);
    - NAND_GATE port map (SB, S2);
end GEN_IN_COMP;
```

Giả sử các thành phần NAND_GATE và AND_GATE được buộc (*bound*) tới các thực thể (*entity*) NAND_GATE và AND_GATE đã mô tả trước đó. Khai báo thành phần (*component declaration*) cho AND_GATE đặc tả một giá trị cho *generic*. Khi thành phần (*component*) này được tạo nấc (*instantiate*) và một giá trị *generic* mới được gán dùng ánh xạ *generic* như trong nấc (*instance*) A1, giá trị mới là 10, dẫn xuất từ giá trị đặc tả trong khai báo thành phần (*component declaration*) là 5. Khi thành phần (*component*) AND_GATE được tạo nấc (*instantiate*) và không có ánh xạ *generic* nào được đặc tả, như trong nấc (*instance*) A2, thì giá trị *generic* đã đạc tả trong khai báo thành phần là 5, được sử dụng. Trong trường hợp của nấc (*instance*) N1, giá trị được đặc tả trong ánh xạ *generic* là 6, được dẫn xuất giá trị đã gán cho *generic* trong khai báo thực thể (*entity declaration*) NAND_GATE là 2. Trường hợp nấc (*instance*) N2 trong dòng chú thích là không hợp lệ, bởi vì không có cả tạo nấc lẫn khai báo cung cáp giá trị cho *generic*.

Phần thân kiến trúc (*architecture body*) đã mô tả trước đó dùng cú pháp tổng quát hơn cho câu lệnh khai báo thành phần và tạo nấc thành phần:

- Khai báo thành phần:

component *component-name* [is]
    [generic(*list-of-generics*);]
    [port(*list-of-interface-ports*);
end component[*component-name*];

- Tạo nấc thành phần:

component-label: *component-name*
    [generic map(*generic-association-list*)]
    [port map(*ports-association-list*)];

Các giá trị cho các *generic* cũng có thể được đặc tả trong phần đặc tả cấu hình (*configuration specification*) hay khai báo cấu hình (*configuration declaration*). Chúng ta sẽ xem xét trong phần cấu hình.

Ví dụ: mô hình của cổng NOR với *generic* tăng giảm trễ (*delays*):

```
entity NOR2 is
    generic(PT_HL,PT_LH:TIME range 0 ns to TIMEHIGH);
```

```
        port(DA, DB: in BIT; DZ: out BIT);
end NOR2;
architecture NOR2_DELAYS of NOR2 is
        signal TEMP: BIT;
begin
        TEMP <= not(DA or DB);
        DZ <= TEMP after PT_HL when TEMP = '0' else
                 TEMP after PT_LH;
end NOR2_DELAYS;
```

Bởi vì không có các giá trị mặc định được cung cấp cho các *generic*, các giá trị phải được cung cấp khi thực thể này được tạo nấc (*instantiate*) hoặc cấu hình (*configure*).

Giả sử một cổng OR được xây dựng dùng 2 cổng NOR; mỗi cổng NOR được mô tả trước. Tăng giảm trễ được đặc tả khi thành phần (*component*) NOR2 được tạo nấc.

Ví dụ: sự lan truyền trễ được đặc tả trong từng câu lệnh tạo nấc thành phần (*component instantiation*).

```
entity OR2 is
        port(A,B: in BIT; C: out BIT);
end OR2;

architecture OR2_NOR2 of OR2 is
        component NOR2
                generic(PT_HL, PT_LH: TIME);
                port(A, B: in BIT; Z: out BIT);
        end component;
        signal S1: BIT;
begin
        N1:NOR2 generic map(5 ns,3 ns) poprt map(A,B,S1);
        N2:NOR2 generic map(6 ns,
                                        5 ns) poprt map(S1,S1,C);
end;
```

Các công dụng khác của các *generic* bao gồm các lĩnh vực xây dựng mô hình (*modeling ranges*) của các kiểu con (*subtype*)

Ví dụ:

```
subtype ALUBUS is INTEGER range TOP downto 0
```

- TOP là một *generic*.

Các *generic* cũng có thể được dùng để điều khiển số lượng nấc (*instance*) của một thành phần (*component*) trong phát biểu tổng quát (xem chương 10).

7.2. TẠI SAO ĐỊNH CẤU HÌNH (*configuration*)?

Đây là hai lý do chính:

1. Đôi khi nó có thể tiện lợi để đặc tả nhiều phép chiếu (*view*) cho một thực thể đơn (*single entity*) và sử dụng một trong các phép chiếu này để mô phỏng. Điều này có thể được thực hiện dễ dàng bằng các đặc tả một thân kiến trúc (*architecture body*) cho từng phép chiếu (*view*) và dùng một cấu hình (*configuration*) để liên kết (*bind*) thân kiến trúc (*architecture body*) đã mô tả. Ví dụ, ở đây có thể có ba thân kiến trúc (*architecture body*), gọi là FA_BEH, FA_STR, và FA_MIXED, tương ứng với một thực thể (*entity*) FULL_ADDER. Một kiến trúc (*architecture*) bất kỳ có thể được chọn để mô phỏng bằng cách đặc tả một cấu hình (*configuration*) thích hợp.
2. Tương tự trường hợp trên, có thể mô tả bằng cách kết nối (*associate*) một thành phần (*component*) với một tập thực thể (*entity*) bất kỳ. Khai báo thành phần (*component declaration*) có thể gồm có tên thành phần (*component*) và tên, kiểu, số lượng cổng và các *generic* khác từ các cổng này của thực thể của nó.
   Ví dụ: khai báo một thành phần (*component*) sử dụng trong thiết kế:
   ```
   component OR2
   ```

```
port(A,B: in BIT; Z: out BIT);
end component;
```

và các thực thể mà thành phần trên có thể bị buộc (*bound*) là:

```
entity OR_GENERIC is
        port(N: out BIT; L,M: in BIT);
end OR_GENERIC;
entity OR_HS is
        Port(X,Y: in BIT; Z: out BIT);
end OR_HS;
```

Tên thành phần, tên thực thể, cũng như tên cổng và thứ tự của chúng là khác nhau. Trong trường hợp chúng ta có thể dùng thực thể (*entity*) OR_HS cho thành phần (*component*) OR2, và trong trường hợp khác, là thực thể OR_GENERIC. Điều này có thể được thực hiện bằng các đặc tả một cấu hình phù hợp cho thành phần (*component*). Lợi thế là ở chỗ khi các thành phần được sử dụng trong thiết kế, cho các thành phần và các cổng giao tiếp của chúng có thể được sử dụng tên tùy ý, và sau đó có thể được buộc (*bound*) tới các thực thể (*entity*) trước khi mô phỏng (*simulation*).

Vì vậy một cấu hình được dùng để nối từng cặp như sau:

- Một thân kiến trúc (*architecture body*) tới khai báo thực thể (*entity declaration*) của nó.
- Một thành phần (*component*) với một thực thể (*entity*).

Chú ý: một cấu hình không có bất kỳ một ngữ nghĩa mô phỏng nào kết hợp với nó; nó chỉ đặc tả một thực thể mức cao nhất (*top-level entity*) được tổ chức như thế nào trong giới hạn (*term*) của các thực thể mức thấp hơn (*lower-level entities*) bằng cách đặc tả sự ràng buộc (*binding*) giữa các thực thể. Ngôn ngữ cung cấp hai cách để biểu diễn sự ràng buộc này:

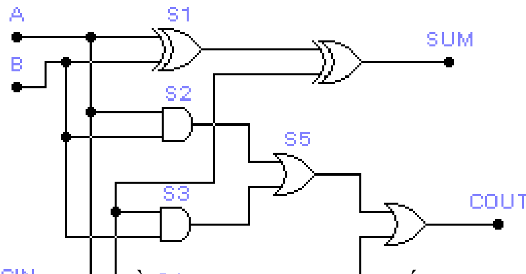
1. Bằng cách dùng đặc tả cấu hình
2. Bằng cách dùng khai báo cấu hình

7.3. ĐặC Tả CấU HÌNH (*configuration specification*)

Một đặc tả cấu hình được sử dụng để ràng buộc (*bind*) các nấc thành phần (*component instantiation*) với các thực thể riêng biệt (*specific entities*) lưu trữ trong các thư viện thiết kế. Việc đặc tả xuất hiện trong phần khai báo của kiến trúc (*architecture*) hoặc trong khối các thành phần được tạo nấc. Việc ràng buộc một thành phần với một thực thể có thể được thực hiện trên một *perinstance* cơ bản, cho tất cả các nấc (*instantiation*) của một thành phần thành phần (*component*), hoặc cho một tập hợp các nấc (*instantiation*) của một thành phần thành phần (*component*) đã được chọn lựa. Các nấc của các thành phần khác nhau cũng có thể được ràng buộc (*bound*) với cùng một thực thể thực thể (*entity*).

```
library HS_LIB,CMOS_LIB;
entity FULL_ADDER is
        port(A,B,CIN: in BIT; SUM,COUT: out BIT);
end;
architecture FA_STR of FULL_ADDER is
        component XOR2
                port(D1,D2: in BIT; DZ:out BIT);
        end component;
        component AND2
                port(Z:out BIT; A0,A1: in BIT);
        end component;
        - Đặc tả cấu hình:
        for X1,X2: XOR2
                use entity WORK.XOR2(XO2BEH); - liên kết thực thể
                        - với nhiều nấc của một thành phần
        for A3: AND2
                use entity HS_LIB.AND2HS(AND2STR);
                        port map (HS_B=>A1,HS_Z=>Z,HS_A=>A0);
                        - liên kết thực thể với tạo nấc đơn của một thành phần
        for all: OR2
```

```
                    use entity CMOS_LIB.OR2CMOS(OR2STR); - liên kết
                              - thực thể với tất cả các nấc của thành phần OR2
          for others: AND2
                    use entity WORK.A_GATE(A_GATE_BODY);
                              port map (A0,A1,Z); - liên kết thực thể với tất
                              - cả các nấc không được buộc của thành phần AND2
          signal S1,S2,S3,S4,S5: BIT;
begin
          X1:XOR2 port map (A,B,S1);
          X2:XOR2 port map (S1,CIN,SUM);
          A1:AND2 port map (S2,A,B);
          A2:AND2 port map (S3,B,CIN);
          A3:AND2 port map (S4,A,CIN);
          O1:OR2 port map (S2,S3,S5);
          O2:OR2 port map (S3,B,CIN);
          - NAND_GATE port map (S4,S5,COUT);
end FA_STR;
```



Hình 7.1: Mạch cộng toàn phần 1*bit*

Có bốn đặc tả cấu hình trong phần khai báo của thân kiến trúc (*architecture body*):

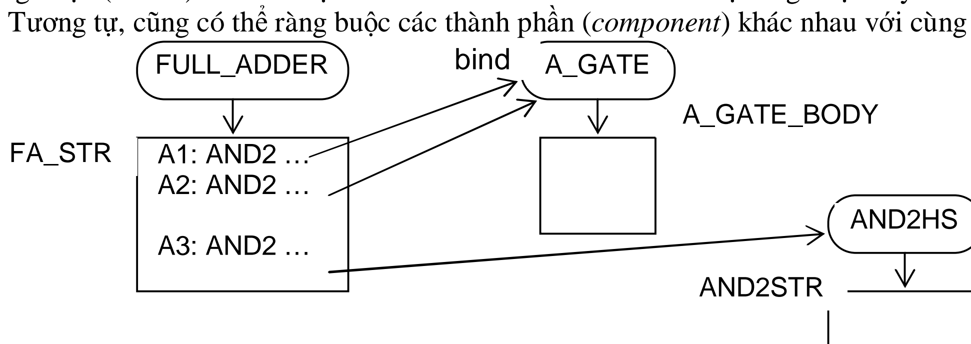
- Đặc tả thứ nhất chỉ rằng nấc (*instance*) X1 và X2 của thành phần thành phần (*component*) XOR2 được ràng buộc với thực thể đại diện bởi cặp thực thể – kiến trúc (*entity-architecture*) XOR2 và XOR2BEH trong thư viện WORK.
- Đặc tả thứ hai ràng buộc (*bind*) nấc A3 của thành phần AND2 với thực thể đại diện bởi cặp thực thể – kiến trúc (*entity-architecture*) AND2HS và AND2STR trong thư viện thiết kế HS_LIB. Ánh xạ của các cổng thành phần (AND2) và các cổng thực thể (AND2HS) được đặc tả dùng liên kết tên

  Ví dụ, cổng HS_A của thực thể AND2HS được ánh xạ tới cổng A0 của thành phần AND2.

- Đặc tả thứ ba chỉ rằng tất cả các nấc (*instances*) của thành phần OR2 sử dụng thực thể đại diện bởi cặp thực thể – kiến trúc (*entity-architecture*) trong thư viện thiết kế CMOS_LIB.
- Đặc tả cuối cùng chỉ rằng tất cả các nấc (*instance*) không ràng buộc (*unbound*) của thành phần thành phần (*component*) AND2, nghĩa là nấc (*instance*) A1 và A2, được ràng buộc tới thực thể (*entity*) A_GATE dùng kiến trúc (*architecture*) A_GATE_BODY trong thư viện WORK.

Trong ví dụ trước đã minh họa rằng các nấc (*instance*) của cùng một thành phần (*component*) có thể được ràng buộc (*bound*) với các thực thể khác nhau. Hình 7.2 mô tả sự ràng buộc này.

Tương tự, cũng có thể ràng buộc các thành phần (*component*) khác nhau với cùng một thực thể



Hình 7.2: Các nấc khác nhau buộc tới các thực thể khác nhau

(*entity*). Xem ví dụ trong hình 7.3. Hình này mô tả rằng không có gì đặc biệt trong thành phần (*component*) AND2. Bằng cách ràng buộc một nấc (*instance*) của thành phần (*component*) AND2 với thực thể (*entity*) OR_GATE, nấc này được xây dựng như là đặc tả trong kiến trúc của thực thể (*entity*) OR_GATE. Mối liên kết (*bind*) như vậy có thể làm cho người đọc không đồng ý, mặc dù nó đúng, nhưng đôi khi cần thiết. Ví dụ, trong khi chạy từng bước (*debugging*) một mô hình chúng ta có thể muốn thấy ảnh hưởng của việc đặc tả một cổng AND đối với một cổng OR mà không thay đổi các phần mô tả còn lại .Thực thể (*entity*) P có bốn đối tượng thành phần (*component instance*), PX1 và PX2 của thành phần (*component*) kiểu PX, PY1 của thành phần kiểu PY và PW1 của thành phần kiểu PW. Các đối tượng thành phần PX1 và PY1 được buộc (*bound*) tới thực thể Q, trong khi PX2 được buộc tới R. Các đối tượng thành phần QL1 và QL2 (của thành phần kiểu QL) trong thực thể Q được buộc tới các thực thể S và T. Các đối tượng thành phần RL1 và RM1 trong thực thể R được buộc tới các thực thể S và W. Tất cả các đối tượng thành phần trong S và T, và thành phần PW1 trong thực thể P, được buộc tới thực thể đơn W. Nói cách khác, tất cả các thực thể P, Q, R, S, và T được xây dựng kế thừa (*hierarchically*) từ một thành phần đơn ban đầu là W.

Cú pháp đặc tả cấu hình:

for *list-of-comp-labels*: *component-name binding-indication*;

binding-indication: thực thể đại diện bởi cặp thực thể kiến trúc, và các mối liên kết cổng và *generic*, và đây là một dạng của nó:

use entity *entity-name* [(*architecture-name*)]
[generic map (*generics-association-list*)]
[port map (*port-association-list*)   - form 1

Danh sách nhãn các thành phần có thể được thay thế bằng từ khóa chỉ rõ tất cả các nấc (*instance*) của một thành phần (*component*), nó cũng có thể là các từ khóa khác đặc tả tất cả các nấc (*instance*) chưa được buộc của một thành phần. Ánh xạ *generic* (*generic map*) được sử dụng để đặc tả các giá trị cho *generic* hoặc cung cấp ánh xạ giữa các tham số *generic* của thành phần (*component*) và thực thể (*entity*) mà nó buộc tới. Ánh xạ cổng (*port map*) được sử dụng để đặc tả mối ràng buộc cổng (*binding port*) giữa thành phần và thực thể buộc.

Ví dụ: đặc tả cấu hình thân kiến trúc (*architecture body*):

```
architecture DUMMY of DUMMY is
	component NOR_GATE
		generic(RISE_TIME, FALL_TIME: TIME);
		port(S0,S1: in BIT; Q: out BIT);
	end component;
	component AND2_GATE
		port(DOUT: out BIT; DIN: BIT_VECTOR);
	end component;
	for N1,N2: NOR_GATE
		use entity WORK.NOR2(NO2_DELAYS);
			generic map (PT_HL => FALL_TIME,
					PT_LH => RISE_TIME)
			Port map (S0,S1,Q);
	for all: AND2_GATE
		use entity WORK.AND2(GENERIC_EX);
			generic map (10)
			port map (A => DIN, Z => DOUT);
	signal WR,RD,RW,S1,S2: BIT;
	signal SA: BIT_VECTOR(1 to 10);
begin
	N1:NOR_GATE generic map(2 ns,3 ns) poprt map(WR,RD,RW);
```

```
        A1:AND2_GATE poprt map (S1,SA);
        N1:NOR_GATE generic map(4 ns,6 ns)
                                poprt map(S1,SA(2),S3);
end DUMMY;
```

Các khai báo **entity** cho các thực thể được buộc tới các thành phần NOR_GATE và AND2_GATE là:

```
entity NOR2 is
        generic(PT_HL,PT_LH: TIME);
        port(DA,DB: in BIT; DZ: out BIT);
end NOR2;
entity AND2 is
        generic(N:NATURAL := 5);
        port(A: in BIT_VECTOR(1 to N); Z: out BIT);
end AND2;
```

Trong mối liên kết cho N1 và N2, ánh xạ *generic* ánh xạ các tên *generic* từ thực thể (*entity*) NOR2 đến thành phần (*component*) NOR_GATE dùng liên kết tên. Các giá trị *generic* đã cung cấp trong tạo nấc (*instantiation*) vì vậy được gửi đến thực thể (*entity*) NOR2 qua ánh xạ này. Ràng buộc cổng được đặc tả dùng liên kết vị trí; nghĩa là, cổng S0, S1, Q của thành phần (*component*) NOR_GATE ánh xạ đến các cổng tương ứng DA, DB và DZ của thực thể NOR2. Đặc tả cấu hình cho AND2_GATE chỉ định giá trị 10 cho *generic* (dùng liên kết vị trí), dẫn xuất từ giá trị mặc định (5) đã chỉ định trong khai báo **entity** của thực thể AND2. Khai báo **component** cho AND2_GATE sẽ không đặc tả bất kỳ một *generics* nào, bởi vì các giá trị được gửi trực tiếp đến các *generics* của thực thể. Ánh xạ cổng cho AND2_GATE được đặc tả dùng liên kết tên.

Làm thế nào để các giá trị ánh xạ cổng và ánh xạ *generic* trong tạo nấc thành phần (*component instantiation*) gửi vào thực thể ràng buộc với nó theo đặc tả cấu hình? Hãy lấy N1 trong thân kiến trúc (*architecture body*) trước làm ví dụ:

```
N1: block – khối cho tạo nấc thành phần
        generic(RISE_TIME,FALL_TIME:TIME);- các generic của thành phần
        generic map (RISE_TIME => 2 ns, FALL_TIME => 3 ns);
        port (S0,S1:in BIT; Q:out BIT); -các cổng của thành phần
        port map (S0 => WR, S1 => RD, Q => RW); - ánh xạ cổng trong tạo nấc
begin
        NOR2: block
                generic (PT_HL,PT_LH:TIME); -các generic của nó
                generic map (PT_HL=>FALL_TIME, PT_LH=>RISE_TIME);
                                        - ánh xạ generic trong đặc tả cấu hình
                port (DA,DB:in BIT; DZ:out BIT); -các cổng của nó
                port map (DA => S0, DB => S1, DZ => Q); - ánh xạ cổng trong đặc tả
                - các khai báo khác trong thân kiến trúc NOR2_DELAYS
        begin
                - các phát biểu trong thân kiến trúc NOR2_DELAYS
        end block N1;
```

Khối N1 được tạo thành từ tạo nấc thành phần (*component instantiation*) của NOR_GATE. Ánh xạ *generic* và ánh xạ cổng cho khối này là ánh xạ *generic* và ánh xạ cổng đặc tả trong tạo nấc thành phần (*component instantiation*) cho N1; nghĩa là, chúng đặc tả ánh xạ giữa các giá trị và tín hiệu trong phát tạo nấc thành phần (*component instantiation*) với các *generics* và các cổng của thành phần (*component*) NOR_GATE. Bên trong khối NOR2 mô tả thực thể (*entity*) NOR2 mà tạo nấc thành phần (*component instantiation*) N1 được ràng buộc tới trong đặc tả cấu hình. Ánh xạ *generic* và ánh xạ cổng của khối này đặc tả ánh xạ *generic* và ánh xạ cổng trong đặc tả cấu hình; nghĩa là, chúng đặc tả ánh xạ giữa thành phần (*component*) NOR_GATE và thực thể (*entity*) NOR2.

7.4. KHAI BÁO CẤU HÌNH (*configuration declaration*)

Khai báo cấu hình phải xuất hiện trong một thân kiến trúc (*architecture body*). Vì vậy, để thay đổi mối liên kết (*binding*) ta phải thay đổi thân kiến trúc và phân tích lại. Điều này có thể gây trở ngại và tốn thời gian. Đẽ tránh điều này, người ta dùng khai báo cấu hình để đặc tả một mối liên kết.

Một khai báo cấu hình là một đơn vị thiết kế độc lập. Vì vậy nó cho phép liên kết trễ các thành phần, nghĩa là, các mối liên kết có thể được thực hiện sau khi đã viết thân kiến trúc (*architecture body*). Nó cũng có thể có nhiều khai báo cấu hình cho một thực thể (*entity*), mỗi khai báo cấu hình định nghĩa một tập hợp các mối liên kết cho các thành phần trong một thân kiến trúc đơn (*single architecture body*), hoặc có thể đặc tả một cặp đơn vị thực thể – kiến trúc (*unique entity-architecture pair*).

Dạng thức đặc trưng của khai báo cấu hình là:

configuaration *configuaration-name* of *entity-name* is
block-configuaration
end [configuaration] [*configuaration-name*];

Nó khai báo một cấu hình (*configuration*) với tên là *configuration-name* của thực thể *entity-name*. Một *block-configuration* định nghĩa mối liên kết của các thành phần trong khối, nơi mà một khối có thể là một thân kiến trúc (*architecture body*), một phát biểu khối, hoặc một phát biểu phát sinh (*generate*). Mối liên kết (*binding*) của các thành phần định nghĩa trong một phát biểu khối và trong một phát biểu phát sinh (*generate*) được thảo luận trong chương 10. Cấu hình khối là một cấu trúc đệ quy có dạng:

for *block-name* of *entity-name* is
component-configuarations
block-configuarations
end for;

block-name là tên của một thân kiến trúc (*architecture body*), nhãn của một phát biểu khối, hoặc nhãn của một phát biểu phát sinh. Khối mức cao nhất (*top-level block*) luôn luôn là một thân kiến trúc. Một *component-configuration* liên kết các thành phần trong một khối với các thực thể, và có dạng:

for *list-of-comp-labels*: *comp-name*[*binding-indication*;]
[*block-configuarations*]
end for;

Cấu hình khối bên trong một cấu hình thành phần định nghĩa mối liên kết của các thành phần ở mức kế tiếp trong cặp thực thể – kiến trúc (*entity-architecture*) đặc tả bởi dấu hiệu liên kết (*binding indication*).

Có hai dạng khác của dấu hiệu liên kết:

use configuration *configuration-name* - Dạng 2
use open - Dạng 3

Trong dạng 2, dấu hiệu liên kết (*binding indication*)chỉ rằng các nấc thành phần (*component instance*) được buộc tới một cấu hình của một thực thể (*entity*) mức thấp hơn chỉ bởi tên cấu hình. Nghĩa là khai báo cấu hình với một tên như vậy phải tồn tại. Trong dạng 3, dấu hiệu liên kết (*binding indication*) chỉ rằng các mối liên kết chưa được đặc tả và bị trì hoãn lại. Cả hai dạng này của dấu hiệu liên kết cũng có thể dùng trong đặc tả cấu hình.

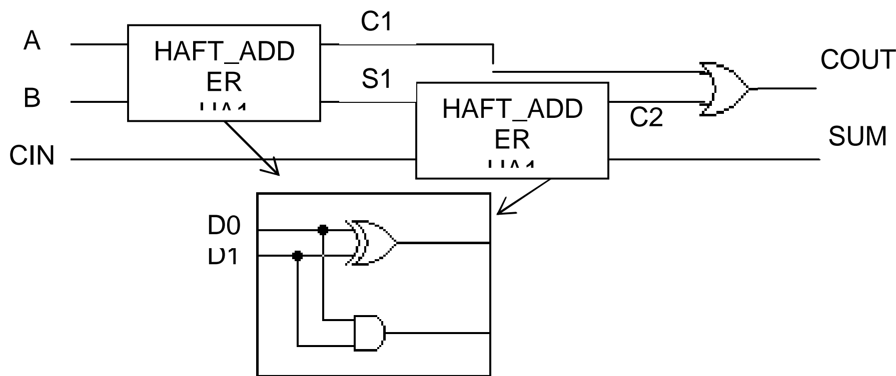
Ví dụ: khai báo cấu hình đặc tả cấu hình thành phần cho tất cả các nấc thành phần (*component instances*) trong kiến trúc (*architecture*) FA_STR của thực thể (*entity*) FULL_ADDER đã mô tả trong phần trước:

```
library CMOS_LIB;
configuration FA_CON of FULL_ADDER is
    for FA_STR
        use WORK.all;
        for A1,A2,A3: AND2
            use entity CMOS_LIB.BIGAND2(AND2STR);
        end for;
    for others: OR2 –dùng mặc định, dùng OR2 từ thư viện WORK
    end for;
    for all: XOR2
        use configuration WORK.XOR2CON;
    end for;
```

```
        end for;
    end FA_CON;
```

Cấu hình (*configuration*) FA_CON liên kết kiến trúc (*architecture*) FA_STR với thực thể (*entity*) FULL_ADDER. Các thành phần bên trong thân kiến trúc (*architecture body*) này, nấc (*instance*) A1, A2 và A3, được buộc tới thực thể (*entity*) BIGAND2 trong thư viện CMOS_LIB. Với tất cả các nấc (*instance*) của thành phần (*component*) OR2, các mối liên kết mặc định được sử dụng; đây là các thực thể trong thư viện cùng tên với tên thành phần. Cấu hình thành phần cuối cùng mô tả một kiểu khác của dấu hiệu liên kết (*binding indication*). Trong trường hợp này, tất cả các nấc thành phần (*component instance*) được buộc tới một cấu hình thay vì cặp thực thể – kiến trúc (*entity-architecture*). Tất cả các nấc (*instance*) của thành phần (*component*) XOR2 được buộc tới cấu hình (*configuration*) XOR2CON trong thư viện làm việc. Kiểu liên kết này cũng có thể được đặc tả trong đặc tả cấu hình.

Điểm mạnh của khai báo cấu hình là ở chỗ các thành phần con (*sub-component*) có thể được buộc dùng khai báo cấu hình đơn. Ví dụ, giả sử một mạch cộng toàn phần (*full-adder circuit*) gồm hai bộ cộng bán phần (*full-adder*) và một cổng OR. Mạch cộng bán phần bao gồm một cổng XOR và AND. Sự phân cấp cửa bộ cộng toàn phần được minh họa trong hình 7.5



Hình 7.5: Bộ cộng toàn phần 1*bit* phân cấp

Mô hình cấu trúc cho mạch cộng toàn phần và mạch cộng bán phần, khai báo cấu hình đặc tả mối liên kết cho các thành phần trong bộ cộng toàn phần:

```
    entity FULL_ADDER is
        port(A,B,CIN: in BIT; SUM,COUT: out BIT);
    end;

    architecture FA_WITH_HA of FULL_ADDER is
        component HAFT_ADDER
            port(HA,HB: in BIT; HS,HC:out BIT);
        end component;
        component OR2
            port(A,B:in BIT; Z: out BIT);
        end component;
        signal S1,C1,C2: BIT;
    begin
        HA1:HAFT_ADDER port map (A,B,S1,C1);
        HA2:HAFT_ADDER port map (S1,CIN,SUM,C2);
        O1:OR2 port map (C1,C2,COUT);
    end FA_WITH_HA;
    entity HA is
        port(D0,D1: in BIT; S,C: out BIT);
    end HA;
```

```
architecture HA_STR of HA is
	component XOR2
		port(X,Y: in BIT; Z:out BIT);
	end component;
	component AND2
		port(L,M:in BIT; N: out BIT);
	end component;
begin
	X1:XOR2 port map (D0,D1,S);
	A1:AND2 port map (D0,D1,C);
end HA_STR;
library ECL;
configuration FA_HA_CON of FULL_ADDER is
	for FA_WITH_HA: - cấu hình khối mức cao nhất
		for HA1,HA2: HAFT_ADDER
			use entity WORK.HA(HA_STR);
			   port map(D0=>HA,D1=>HB,S=>HS,C=>HC);
			for HA_STR – cấu hình khối lồng nhau
				for all: XOR2
					use entity WORK.XOR2(XOR2);
				end for;
			for A1:AND2
				use configuration ECL.AND2CON;
			end for;
		end for;
		for O1:OR2
			use configuration WORK.OR2CON;
		end for;
	end for;
end FA_HA_CON;
```

Cấu hình khối mức cao nhất đặc tả mối liên kết của các *component instances* trình bày trong thân kiến trúc (*architecture body*) FA_WITH_HA. Nấc (*instance*) HA1 và HA2 được buộc tới thực thể đặc tả bởi cặp thực thể – kiến trúc (*entity-architecture*)**,** thực thể (*entity*) HA và kiến trúc (*architecture*) HA_STR. Cấu hình khối lồng nhau đặc tả mối liên kết của các nác thành phần (*component instance*) trình bày trong thân kiến trúc (*architecture body*) HA_STR. Bằng cách này, một cấu hình có thể được lồng vào nhau với độ sâu tùy ý và có thể được dùng để liên kết tất cả các thành phần trong một phân cấp.

Theo như ví dụ trên, khi các thành phần trong một phân cấp được ràng buộc, một khai báo cấu hình đơn có thể được sử dụng để thay thế một tập hợp đặc tả cấu hình. Nếu các đặc tả cấu hình đã được sử dụng trong ví dụ trước, chúng phải được bao gồm một cách độc lập trong thân kiến trúc (*architecture body*) FA_WITH_HA và HA_STR, và các thân kiến trúc này sẽ phải được biên dịch lại mỗi khi thay đổi mối liên kết. Chú ý, mỗi nác thành phần (*component instance*) không được buộc đồng thời với đặc tả cấu hình.

7.5. QUY TẮC MẶC ĐỊNH (*default rules*)

Một số lượng lớn thủ tục bổ sung được giới thiệu nếu các mối liên kết cho mỗi thành phần trong thiết kế được đặc tả độc lập. Một cách tiện lợi, ngôn ngữ cung cấp các quy tắc liên kết mặc định. Với mỗi nác thành phần (*component instance*) không ràng buộc:

- Một thực thể khả kiến và cùng tên với thành phần được dùng để liên kết với nác (*instance*); nếu không tồn tại một thực thể như vậy, một dấu hiệu liên kết (*binding indication*) mặc định của “use open” được sử dụng.
- Phần lớn cấu trúc đã phân tích mới nhất cho thực thể được sử dụng; có lỗi nếu không có một kiến trúc (*architecture*) nào tồn tại.

- Với mỗi cổng hoặc *generic* trong nác thành phần (*component instance*), phải tồn tại một cổng hoặc *generic* tương ứng trong thực thể (*entity*) có tên, kiểu (*type*) và chế độ (*mode*) phù hợp; bất kỳ các cổng hoặc *generic* nào trong thực thể mà không kiên kết được xem như mở (*open*); nếu thiếu, nó là một lỗi.

Các quy tắc mặc định giúp tránh các mối liên kết đặc biệ trong các trường hợp tên thành phần trùng với tên thực thể, một tiện lợi khác nữa là nó cho phép dùng tên thành phần chuẩn như là SN7400 và SN7402. Trong các trường hợp này, không có kiên kết nào kà cần thiết. Tuy nhiên, một cấu hình tối thiểu đôi khi cũng có thể cần thiết. Nó có thể có dạng:

```
library TTL_LIB;
configuration TLC_CON of TLC is
        for TLC_STRUCTURE
                use TTL_LIB.all;
        end for;
end TLC_CON;
```

TLC là một thực thể đôi khi tạo nấc thành phần (*component instantiation*) đã định nghĩa bên trong thân kiến trúc (*architecture body*) của nó. Câu lệnh "use TTL_LIB.all" làm cho tất cả các thực thể trong thư viện TTL_LIB khả kiến. Vì vậy, các thực thể này sẽ buộc tới các nấc thành phần trong thân thân kiến trúc bằng quy tắc mặc định.

7.6. HÀM CHUYỂN ĐỔI (*conversion functions*)

Có thể chuyển đổi kiểu cổng của một thực thể thành kiểu khác từ kiểu cổng của thành phần đang buộc tới. Ví dụ, cho khai báo thành phần:

```
component COUNTER
        port (CTR: inout MVL_VECTOR;
                CLK,RST: in RST; PAR: out MVL);
end component;
```

thành phần COUNTER được buộc tới một thực thể với khai báo:

```
entity COUNTER is
        port (Q: inout STD_LOGIC_VECTOR(3 downto 0);
                CLOCK,RESET: in STD_LOGIC;
                PARITY: out STD_LOGIC);
end COUNTER;
```

Điều này xảy ra khá thường xuyên trong thực tế, ví dụ, khi một người thiết kế cung cấp một thư viện khai báo thực thể dùng kiểu ưa thích của mình, trong khi người sử dụng xây dựng mô hình thiết kế dùng kiểu dữ liệu khác. Trong trường hợp như vậy, hàm chuyển đổi có thể được dùng để đổi các giá trị từ kiểu này sang kiểu khác bằng cách đặc tả hàm trong danh sách kết hợp các mối liên kết.

Trong thông tin liên kết của một đặc tả cấu hình, khi dòng dữ liệu từ một thành phần tới một thực thể, nghĩa là, với một cổng nhập, hàm chuyển đổi được đặc tả với tên cổng thành phần; nghĩa là hàm chuyển đổi được thêm vào trên giá trị của cổng thành phần trước khi gán nó cho cổng thực thể. Với một cổng xuất hàm chuyển đổi được đặc tả với tên cổng thực thể, bởi vì dòng dữ liệu từ thực thể đến thành phần, trong khi với một cổng nhập xuất (*inout port*), hàm chuyển đổi được đặc tả trên cả hai cổng thành phần và cổng thực thể.

Ví dụ:
```
for all: COUNTER use entity WORK.COUNTER
        port map(
                TO_MVLVECTOR(Q) => TO_STDLOGICVECTOR(CTR),
                CLOCK => TO_STDLOGIC(CLK),
                RESET => TO_STDLOGIC(RST),
                TO_MVL(PARITY) => PAR);
```

CLOCK và RESET là các cổng nhập của thực thể. Vì vậy, dòng dữ liệu từ các cổng thành phần (CLK và RST) đến các cổng thực thể. Trong trường hợp này, hàm chuyển đổi TO_STDLOGIC xuất hiện trên các cổng thành phần. Giá trị của cổng thành phần là tham số gửi tới hàm chuyển đổi. Giá trị trả về từ hàm chuyển đổi được gửi tới cổng thực thể.

Với ngõ xuất PARITY, hàm chuyển đổi được đặc tả với cổng thực thể. Giá trị cổng thực thể được gửi như là một tham số tới hàm chuyển đổi TO_MVL, và giá trị trả về được gửi tới cổng thành phần PAR.

Với cổng nhập xuất Q, hàm chuyển đổi xuất hiện đồng thời trên cổng thành phần và cổng thực thể, bởi vì dữ liệu di chuyển theo cả hai đường. Khi dữ liệu đi từ thành phần đến thực thể, hàm chuyển đổi TO_STDLOGICVECTOR được gọi, và giá trị trả về được gửi tới cổng thực thể. Nếu dữ liệu đi từ thực thể tới thành phần, hàm chuyển đổi TO_MVLVECTOR được gọi, và giá trị trả về được gửi tới cổng thành phần CTR.

Hàm chuyển đổi được sử dụng ở bất kỳ nơi nào có sự kết hợp các *generic*, các cổng, hoặc các tham số, như sự kết hợp tham số trong một hàm hay một thủ tục gọi, trong ánh xạ cổng và ánh xạ *generic* của một phát biểu khối, hoặc một phát biểu tạo nác thành phần (*component instantiation*).

7.7. TẠO NẮC TRỰC TIẾP (*direct instantiation*)

Một khai báo thành phần khai báo giao diện của một thành phần. Thành phần này có thể được tạo nấc dùng phát biểu tạo nác thành phần. Tuy nhiên, trước khi thực thể chứa thành phần có thể được mô phỏng, nấc thành phần (*component instance*) cần phải được buộc hoặc liên kết đến một cặp thực thể - kiến trúc *(entity-architecture*) hoặc tới một cấu hình. Mối ràng buộc này được đặc tả dùng cấu trúc bổ sung, nghĩa là, dùng đặc tả cấu hình hoặc khai báo cấu hình.

Tuy nhiên, có thể tạo nấc (*instantiate*) trực tiếp cặp thực thể - kiến trúc (*entity-architecture*) hoặc một cấu hình trong một phát biểu tạo nấc thành phần (*component instantiation statement*). Điều này lưu trữ bước liên kết bổ sung khi dùng các thành phần (*component*). Đây là hai dạng bổ sung của phát biểu tạo nấc thành phần có thể được sử dụng để tạo nấc trực tiếp một thực thể hoặc một cấu hình:

component-label: entity *entity-name*[(*architecture-name*)]
    [generic map (*generic-association-list*)]
    [port map (*port-association-list*)];

component-label: configuration *configuration-name*
    [generic map (*generic-association-list*)]
    [port map (*port-association-list*)];

Ví dụ: mô tả của bộ cộng toàn phần 1*bit* phân cấp dùng tạo nấc trực tiếp. Chú ý, trong trường hợp này có thể không hoặc cần thiết khai báo cấu hình, bởi vì các nấc thành phần (*component instantiations*) tạo nấc trực tiếp các cặp thực thể-kiến trúc hoặc các cấu hình thích hợp. Cũng như vậy, có thể hoặc không cần thiết khai báo thành phần.

```
entity FULL_ADDER is
        port(A,B,CIN: in BIT; SUM,COUT: out BIT);
end FULL_ADDER;
architecture FA_WITH_HA of FULL_ADDER is
        signal S1,C1,C2: BIT;
begin
        HA1:entity WORK.HA(HA_STR) port map (A,B,S1,C1);
        HA2: entity WORK.HA(HA_STR) port map (S1,CIN,
                                                        SUM,C2);
        O1:configuration WORK.OR2CON port map (C1,C2,
                                                        COUT);
end FA_WITH_HA;
entity HA is
        port(D0,D1: in BIT; S,C: out BIT);
end HA;

library ECL;
architecture HA_STR of HA is
begin
        X1:entity WORK.XOR2(XOR2) port map (D0,D1,S);
        A1: configuration ECL.AND2CON port map (D0,D1,C);
end HA_STR;
```

Ví dụ: bộ cộng toàn phần với kiến trúc (*architecture*) FA_STR, đã mô tả trong phần 7.3. Chú ý, trong trường hợp này không cần đặc tả cấu hình và khai báo thành phần, bằng cách đó làm cho việc mô tả rất cô đọng.

```
library HS_LIB,CMOS_LIB;
entity FULL_ADDER is
    port(A,B,CIN: in BIT; SUM,COUT: out BIT);
end FULL_ADDER;
architecture FA_STR of FULL_ADDER is
    signal S1,S2,S3,S4,S5: BIT;
begin
    X1:entity WORK.XOR2(XOR2BEH) port map (A,B,S1);
    X2:entity WORK.XOR2(XOR2BEH) port map(S1,CIN, SUM);
    A1:entity WORK.A_GATE(A_GATE_BODY) port map(
                                        S2,A,B);
    A2: entity WORK.A_GATE(A_GATE_BODY) port map(
                                 S1,CIN,SUM,C2);
    A3: entity HS_LIB.AND2HS(AND2STR) port map(
                                     S4,A,CIN);
    O1: entity CMOS_LIB.OR2CMOS(OR2STR) port map (
                                     S4,S5,COUT);
end FA_STR;
```

Có thể làm trễ các mối liên kết của các cổng và các *generic* và các *generic* dẫn xuất (*override prespecified generics*) dùng kỹ thuật liên kết gia tăng (*incremental binding mechnism*). Các mối liên kết (*bindings*) hiện diện trong một đặc tả cấu hình tương ứng với các mối liên kết chính (*primary bindings*). Tuy nhiên, nó có thể không chứa thông tin liên kết hoàn chỉnh. Thiếu thông tin có thể là:

- Các cổng và các *generic* không kết hợp
- Các cổng và các *generic* mở

Ví dụ:

```
entity FULL_ADDER is
    port(A,B,CIN: in BIT; SUM,COUT: out BIT);
end FULL_ADDER;
architecture FA_STR_INCR of FULL_ADDER is
    component XOR2
        port(D1,D2:in BIT; DZ:out BIT);
    end component;
    component AND2
        port(Z:out BIT; A0,A1:in BIT);
    end component;
    component OR2
        port(N1,N2:in BIT; Z:out BIT);
    end component;
    for X1,X2:XOR2
        use entity WORK.MY_XOR2; - đặc tả ánh xạ generic và                -
ánh xạ cổng. Bởi vì tên kiến trúc không được                - chỉ định, mặc định là kiến
trúc phân tích mới nhất
    for others:AND2
        use entity WORK.MY_AND2(ARCH_BODY);
        port map (HS_B => A1, HS_A => open);
            - cổng HS_Z không được kết hợp và cổng HS_A mở
    for all:OR2
        use entity WORK.MY_OR2
        generic map (TPHL => 2 ns, TPLH => 3 ns);
```

```
                    - các giá trị generic sẽ được dẫn xuất sau bằng đặc tả trong khai báo cấu
                    hình.
begin
    …
end FA_STR_INCR;
entity MY_XOR2 is
    generic(TPHL,TPLH:TIME);
    port(XA,XB: in BIT; XZ: out BIT);
end MY_XOR2;
entity MY_AND2 is
    port(HS_A,HS_B: in BIT; HS_Z: out BIT);
end MY_AND2;
entity MY_OR2 is
    generic(TPHL,TPLH:TIME);
    port(N1,N2: in BIT; Z: out BIT);
end MY_OR2;
```

Trong các mối liên kết (*binding*) cho thành phần XOR2, thiếu ánh xạ cổng và *generic*. Trong các liên kết (*binding*) thành phần AND2, cổng HS_Z không được kết hợp, và trong các liên kết (*binding*) cho thành phần OR2, các giá trị *generic* được chỉ định. Tuy nhiên các giá trị *generic* đã đặc tả trong khai báo cấu hình có thể dẫn xuất từ các giá trị *generic* đã chỉ định trong đặc tả cấu hình. Những thông tin phụ và thiếu có thể được cung cấp sau trong khai báo cấu hình, có thể tồn tại trong các tập tin khác nhau. Ví dụ khai báo cấu hình thiếu thông tin:

```
configuration FA_INCREMENTAL of FULL_ADDER is
    for FA_STR_INCR
        - cấu hình thành phần đầu tiên:
        for X1,X2:XOR2
            port map (A0,A1,Z)
            generic map(TPHL => 2 ns, TPLH =>5 ns);
        end for
        - cấu hình thành phần thứ hai:
        for all:AND2
            port map (HS_A => ‘1’,HS_Z => Z)
        end for
        - cấu hình thành phần thứ ba:
        for all:OR2
            generic map(TPHL => 4 ns, TPLH =>6 ns);
        end for;
    end for;
end FA_INCREMENTAL;
```

Chương 8 : CÁC CHƯƠNG TRÌNH CON VÀ QUÁ TRÌNH TRÀN

Chương này mô tả 2 dạng chương trình con là : thủ tục (procedure) và hàm (function) . Một hàm có thể sử dụng cho biểu thức , một thủ tục có thể dùng cho các phát biểu tuần tự và đồng thời . Nếu tồn tại nhiều chương trình con cùng tên thì quá trình đó gọi là tràn (overloading) . Chương này hiện thực khái niệm overloading và giải quyết vấn đề này sau cho tốt nhất .

8.1 CHƯƠNG TRÌNH CON (subprogram) :

Một subprogram định nghĩa 1 thuật toán tuần tự , thực thi chính xác 1 bài toán . Có 2 dạng subprogram :

1. Function : thông thường sử dụng việc tính toán giá trị đơn giản , function được thực hiện trong thời gian mô phỏng là zero .
2. Procedure : sử dụng cho việc phân mảnh mô tả hành vi lớn . Procedure có thể trả ề zero hoặc nhiều giá trị . Một procedure có thể có hoặc không thực hiện thời gian mô phỏng zero , phụ thuộc vào nó có phát biểu wait hay không .

Một subprogram được định nghĩa là *subprogram body* . Dạng 1 subprogram body là :

```
subprogram_specification is
        subprogram_item_declarations
begin
        subprogram_statements    -- Same as sequetial_statements.
end [ function | procedure ] [ subprogram_name ];
```

Subprogram_specification : xác định tên của 1 chương trình con và định nghĩa giao diện của nó , đó là định nghĩa tên các tham số hình thức (formal parameter) , lớp của chúng (signal ,variable , file hoặc constant), loại và kiểu của chúng (in , out ,inout). Các tham số của dạng in là chỉ đọc các tham số , chúng không được cập nhật trong thân chương trình con . Các tham số của dạng out chỉ được viết các tham số , giá trị của chúng không được sử dụng nhưng có thể được cập nhật trong thân chương trình con . Các tham số inout có thể đọc được cũng như cập nhật được . Các file không có kiểu , chúng có thể được đọc hoặc viết tuỳ thuộc vào file có đang mở hay không .

Các tham số thực (actual) trong chương trìnhcon gọi (subprogram call) là sử dụng các giá trị đã có đến và từ 1 chương trình con. Chỉ có tín hiệu số thực có thể liên kết với tham số hình thức của lớp tín hiệu. Chỉ biến số thực có thể liên kết với tham số của lớp biến . Chỉ có file mới là số thực cho tham số của file . một biểu thức có thể sử dụng giá trị đã có cho tham số của lớp constant.

Khi các tham số là của lớp biến hoặc hằng số thì đều qua chương trình con bằng giá trị . Các dãy có thể hoặc không thể qua sự tham khảo . Các file phải qua sự tham khảo . Đối với các tín hiệu , sự tham khảo đến tín hiệu , sự điều khiển hoặc cả hai đều qua chương trình con . Điều đó ý nói rằng 1 phép gán từ tín hiệu trong procedure (các tín hiệu không thể gán các giá trị trong function , bởi vì các tham số bị giới hạn của mode in)

, ảnh hưởng sự điều khiển tín hiệu số thực 1 cách tức thời và độc lập hoặc không độc lập với điểm cuối của procedure . Đối với các tín hiệu của 1 mode , không có dạng thuộc tính giá trị tín hiệu ‘ STABLE’ , ‘QUIET’ , ‘DELAYED’ hoặc ‘TRANSACTION ‘ (thuộc tính được bàn luận trong chương 10) , có thể sử dụng trong thân chương trình con .

Kiểu của số thực trong chương trình con gọi phải tương ứng với tham số hình thức . Nếu tham số hình thức thuộc về kiểu không ràng buộc , kích thước của các tham số này được giới hạn từ số thực được đưa vào .

S*ubprogram_item_declarations :* bao gồm 1 tập hợp các khai báo (khai báo kiểu và đối tượng) có thể thu được trong chương trình con . Các khai báo được đưa vào ảnh hưởng đến toàn bộ thời gian chương trình con được gọi . Biến được khởi tạo và tồn tại trong suốt thời gian chương trình con được gọi , nó giống như biến trong phát biểu process , chỉ được đưa vào 1 lần khi bắt đầu mô phỏng và tồn tại cho tới khi quá trình mô phỏng được hoàn thành .

Subprogram_statements : bao gồm các phát biểu tuần tự thực hiện tính toán bằng chương trình con . Phát biểu trả về (return)là phát biểu đặc biệt được cho phép chỉ trong chương trình con . Dạng của nó là :

Return [expression]

Phát biểu return được tạo ra ở cuối chương trình con và trả giá trị về cho chương trình gọi . Tất cả các function đều có phát biểu return cùng với 1 biểu thức , giá trị của biểu thức trong khai báo return là trả về cho chương trình gọi . Đối với các procedure , đối tượng của mode out và inout , các giá trị của chúng được trả về cho chương trình gọi .

Subprogram_name xuất hiện ở cuối thân chương trình con , nếu tồn tại thì phải trùng với tên của function hoặc procedure mà nó xác định .

8.1.1 Các hàm (function);

Các hàm dùng mô tả các thuật toán tuần tự trả về 1 giá trị . Giá trị này trả về cho chương trình gọi bằng phát biểu return , được sử dụng cho các hàm xác định và các hàm có kiểu chuyển biến . Sau đây là ví dụ 1 thân hàm :

```
function  LARGEST ( TOTAL_NO : INTEGER ; SET : PATTERN )
      return REAL is
         --PATTERN is elsewhere defined to be a type of 1-D array of
         -- non_negative floating point value .
      variable     PATTERN_VALUE : REAL :=0;
begin
      for K in SET’range loop
           if  SET(K) > RETURN_VALUE then
                 RETURN_VALUE := SET (K);
           end if;
      end loop;
      return RETURN_VALUE;
end LARGEST;
```

Biến RETURN_VALUE được đưa vào giá trị đầu là 0.0 trong thời gian hàm được gọi . Nó không còn tồn tại sau điểm cuối của hàm .

Cú pháp chung của chương trình con xác định cho thân hàm là :

[pure | impure] function *function_name (parameter_list)*
return *return_type*

Một hàm nguyên mẫu *(Pure function)* là giá trị trả về tương ứng mỗi thời gian mà hàm gọi đến tập hợp số thực tương ứng . *Impure function* là sự trả về giá trị khác với mỗi thời gian mà nó gọi đến tập hợp số thực tương ứng .

Ví dụ NOW là 1 impure function bởi vì nó trả về giá trị khác với khi gọi tại thời điểm khác . Nếu không có từ khóa pure hay impure thì hàm đó mặc nhiên được xem là pure function .

Parameter_list mô tả danh sách của các tham số hình thức cho function . Mode của các tham số này chỉ cho phép là mode in , ngoài ra chỉ có đối tượng constant và tín hiệu mới cho vào các tham số , lớp đối tượng mặc nhiên là constant. Ví dụ trong function LARGEST , TOTAL_NO là constant và giá trị của nó không thể thay đổi trong thân function .

Một ví dụ khác của thân function , xem phần sau . Function này trả về là true nếu xuất hiện cạnh lên trên tín hiệu input . Function này là 1 pure function :

```
pure function VRISE ( signal CLOCK_NAME : BIT )  return BOOLEAN is
begin
        return CLOCK_NAME ='1' and CLOCK_NAME ' event ;
end VRISE ;
```

Sau đây là 2 ví dụ về impure function :

```
impure  function RANDOM (SEED :REAL) return REAL is
      variable NUM : REAL;
      attribute FOREIGN of RANDOM : function is 'NUM = rand (seed)';
begin
      return NUM;
end RANDOM;

impure  function USE (TO_ALLOCATE : POSITIVE) return POSITIVE is
        --ALLOCATE is a shared variable declared elsewhere , but
        --visible to this function .
begin
      ALLOCATE := ALLOCATE + TO_ALLOCATE ;
      Return ALLOCATE ;
end function USE;   -- keywork function after end is optional.
```

Function gọi là 1 biểu thức và có thể sử dụng trong biểu thức lớn . Ví dụ :

```
SUM := SUM+ LARGEST (MAX_COINS, COLLECTION );
```

Function gọi có dạng là :

Function _ name (list_of_actuals)

Các số thực có thể xácđịnh bởi vị trí (số thực đầu tiên tương ứng với số hình thức đầu tiên , số thực thứ hai tương ứng với số hình thức thứ hai và cứ thế tiếp tục) hoặc xác định bằng liên kết (sự liên kết của các số thực và số hình thức xác định 1 cách rõ ràng). Function gọi trong ví dụ trước , viết lại theo dạng liên kết :

```
LARGEST ( SET => COLLECTION, TOTAL_NO => MAX_COINS)
```

Các hàm thông thường cho việc biến đổi kiểu . sau đây là 1 ví dụ của hàm biến đổi 1 giá trị từ kiểu STD_ULOGIC tới giá trị kiểu CHARACTER :

```
function  TO_CHARACTER ( ARG : STD_ULOGIC )
        return CHARACTER is
```

```
begin
    case ARG is
        when 'U'  => return 'U';
        when 'X'  => return 'X';
        when '0'  => return '0';
        when '1'  => return '1';
        when 'Z'  => return 'Z';
        when 'W'  => return 'W';
        when 'L'  => return 'L';
        when 'H'  => return 'H';
        when '-'  => return '-';
      end case;
  end TO_CHARACTER ;
```

Function gọi có dạng :

```
TO_CHARACTER ( STD_ULOGIC('U'))
```

Hàm trả về ký tữ 'U' . Ngoài ra nó còn có khả năng biến đổi kiểu nhiều hơn , sử dụng bởi bảng look_up . Sau đây là hàm TO_CHARACTER mô tả bảng look_up .

```
type  LOOK_UP is array (STD_ULOGIC) of  CHARACTER;
constant TO_CHARACTER : LOOK_UP := ('U' => 'U', 'X' =>'X',
      '1'=>'1','Z' =>'Z','W' =>'W','L' => 'L','H'=>'H','-'=>'-');
```

Sau đây là biểu thức :

```
TO_CHARACTER ( STD_ULOGIC ('U'))
```

Cho ra ký tự 'U' . Chú ý trong cả 2 hệ thống cách gọi cho việc thực hiện biến đổi kiểu là không đổi.

8.1.2 Các thủ tục (procedure):

Các procedure cho phép phân chia hành vi lớn vào nhiều modul . Khác với function , trong procedure có thể trả về giá trị zero hoặc nhiều hơn 1 giá trị , sử dụng các tham số của mode out và inout . Cú pháp của chương trình con cho thân 1 procedure là :

procedure *procedure_name (parameter_list*)

Parameter_list xác định danh sách của các tham số hình thức cho procedure . Các tham số có thể là các constant , các biến , hoặc các tín hiệu và mode của chúng có thể là in , out , hoặc inout . Nếu lớp của các tham số không xác định rõ ràng thì mặc nhiên nó là constant , nếu nó là mode in , còn nó là biến nếu mode của tham số đó là out hoặc inout .

Một ví dụ thân procedure mô tả hành vi của thuật toán logic :

```
type OP_CODE is ( ADD,SUB,MUL,DIV,LT,LE,EQ);
…
procedure ARITH_UNIT (A,B:in INTEGER ; OP :in OP_CODE ;
                Z:out INTEGER ; ZCOMP : out BOOLEAN ) is
begin
    case OP is
        when ADD => Z := A+B;
        when SUB => Z := A-B;
        when MUL => Z := A*B;
        when DIV => Z := A/B;
        when LT => ZCOMP := A<B;
        when LE => ZCOMP := A<=B;
        when EQ => ZCOMP := A=B;
    end case ;
```

```
end  ARITH_UNIT;
```

Ta xem 1 ví dụ khác của thân 1 procedure , procedure này xoay quanh việc xác định tín hiệu vector ARRAY_NAME , bắt đầu từ bit START_BIT tới bit STOP_BIT , bằng giá trị ROTATE_BY . Lớp đối tượng cho tham số ARRAY_NAME là xác định 1 cách rõ ràng . Biến FILL_VALUE ban đầu tự động được gán là '0' cho procedure được gọi .

```
Procedure ROTATE_LEFT
        ( signal ARRAY_NAME : inout BIT_VECTOR;
          START_BIT,STOP_BIT : in NATUAL;
          ROTATE_BY : in POSITIVE ) is
        Variable FILL_VALUE : BIT; --every time the procedure is called,
                                  -- initial value is BIT'LEFT, which is '0'.
begin
     assert STOP_BIT > START_BIT
          report   "STOP_BIT is not greater than START_BIT"
          severity NOTE;
     for MACVAR3 in 1 to ROTATE_BY loop
          FILL_VALUE := ARRAY_NAME (STOP_BIT);
            for MACVAR1 in STOP_BIT downto (START_BIT+1) loop
                 ARRAY_NAME (MACVAR1) <=
                                  ARRAY_NAME (MACVAR1 –1);
            end  loop;
            ARRAY_NAME (START_BIT) <= FILL_VALUE ;
      end  loop;
  end procedure ROTATE_LEFT; -- keyword procedure after end is
                                       -- octional.
```

Các procedure được yêu cầu bởi procedure gọi . Procedure gọi có thể có mỗi phát biểu tuần tự hoặc đồng thời , điều cơ bản đó là phát biểu số thực của procedure gọi là hiện tại. Nếu gọi bên trong phát biểu process hoặc chương trình con khác thì nó là phát biểu procedure gọi tuần tự , nếu không thì nó là phát biểu procedure gọi đồng thời . Cú pháp của phát biểu procedure gọi là :

```
[ label : ] procedure_name ( list_of_actual );
```

Các số thực được xác định là biểu thức , các biến , các tín hiệu hoặc các file , các số thực đó được đưa vào trong procedure và các tên của các đối tượng đó được gởi đến procedure xử lý . các số thực có thể xác định sự liên kết vị trí hoặc liên kết tên . Ví dụ :

```
ARITH_UNIT (D1,D2,ADD,SUM,COMP ); -- positional association.
ARITH_UNIT ( Z => SUM, B=> D2, A=>D1,
             OP=>ADD,ZCOMP => COMP); --named association.
```

Một phát biểu chương trình con gọi tuần tự là thực thi 1 cách tuần tự liên quan đến các phát biểu tuần tự chung quanh nó . Một phát biểu chương trình con gọi đồng thời là thực thi bất cứ lúc nào khi có 1 sự kiện xảy ra trên các tham số là tín hiệu mode in hoặc inout . Chương trình con gọi đồng thời tương đương 1 quá trình xử lý với 1 chương trình con gọi tuần tự và 1 phát biểu wait , chờ cho đến khi có 1 sự kiện trên các tham số tín hiệu của mode in hoặc inout .

Sau đây là 1 ví dụ của chương trình con gọi đồng thời và phát biểu process tương đương với nó :

```
archtecture DUMMY_ARCH of DUMMY is
```

```
--following is a procedure body :
procedure INT_2_VEC ( signal D :out BIT_VECTOR ;
                       START_BIT,STOP_BIT : in NATUAL ;
                      signal VALUE : in INTEGER ) is
    begin
        procedure behavior here.
    end INT_2_VEC;
begin
    --this is an examble of a concurrent procedure call:
    INT_2_VEC (D_ARRAY,START,STOP,SIGNAL_VALUE);
end DUMMY_ARCH;

--the equivalent process statement for the concurrent procedure call is:
process
begin
    INT_2_VEC (D_ARRAY,START,STOP,SIGNAL_VALUE);
    --this  is  now  a sequential procedure call .
    wait on SIGNAL_VALUE;
    --since SIGNAL_VALUE is an input signal.
end process;
```

Một procedure có thể sử dụng như là 1 phát biểu đồng thời hoặc phát biểu tuần tự . Chương trình con gọi đồng thời thường xuyên dùng các process .

```
postponend  procedure INT_2_VEC ( signal D:out BIT_VECTOR ;
        START_BIT,STOP_BIT : in NATUAL;
        Signal VALUE :in INTEGER) is
begin
       --procedure behavior here .
end INT_2_VEC;
```

Ngữ nghĩa của 1 chương trình con gọi đồng thời dùng postponed là tương đương với ngữ nghĩa của phát biểu process tương ứng với nó , được gọi là 1 portponed process .

Thân 1 process có thể có phát biểu wait , trong khi 1 function thì không có . Các function sử dụng việc tính toán các giá trị 1 cách tức thì . Do đó function không cần phát biểu wait trong đó . Một process gọi 1 procedure với phát biểu phát biểu không có danh sách cảm nhận .

Vì procedure có thể có phát biểu wait , cho nên các biến và các hằng được mô tả trong procedure vẫn giữ giá trị của nó qua tho82I gian wait và tồn tại liên tục đến khi procedure kết thúc .

8.1.3 Sự khai báo :

Thân chương trình con có thể xuất hiện trong phần khai báo của khối gọi thực hiện . Điều đó không thích hợp nếu chương trình con được phân chia nhiều thực thể . Trong trường hợp như vậy , thân chương trình con có thể được mô tả 1 vị trí trong thân 1 gói , và khi ấy trong khai báo gói xác định khai báo chương trình con tương ứng . Nếu khai báo này được đưa vào đơn vị thiết kế khác sử dụng mệnh đề use , thì chương trình con có thể sử dụng được trong đơn vị thiết kế này . Một khai báo chương trình con mô tả tên chương trình con và danh sách của các tham số không có mô tả hành vi bên trong chương trình con , mà chỉ mô tả giao tiếp bên ngoài với chương trình con đó . Cú pháp của khai báo này là :

```
Subprogram_specification;
```

Sau đây là 4 ví dụ khai báo procedure và function :

```
procedure ARITH_UNIT (A,B:in INTEGER ; OP :in OP_CODE ;
                    Z:out INTEGER ; ZCOMP : out BOOLEAN ) ;
pure  function VRISE ( signal CLOCK_NAME :BIT) return BOOLEAN;
impure function RANDOM (SEED :REAL) return REAL ;
function LARGEST (TOTAL_NO: INTEGER; SET:PATTERN)
     return REAL ; -- this is a pure function.
```

Một trường hợp khác các khai báo chương trình con cần sự cho phép các chương trình con gọi nhau liên tiếp . ví dụ :

```
procedure  P(…) …
     variable A : …
begin
     …
     A := Q(B) ; -- illegal function call .
     …
end P ;

function Q(…) …
begin
     …
     P(…);
     …
 end Q;
```

Việc gọi function Q trong procedure P là không hợp lý , vì Q chưa được khai báo , điều này có thể sửa lại bằng cách viết khai báo function cho Q trước procedure B .

8.2 LƯỢNG QUÁ TẢI CHƯƠNG TRÌNH CON (subprogram) :

Đôi khi nó có hai hoặc nhiều hơn các chương trình con có cùng tên , điều đó hợp pháp . Trong trường hợp như vậy , tên chương trình con gọi là overloaded , khi đó các chương trình con tương ứng cũng được gọi là overloaded . Ví dụ có 2 khai báo sau :

```
function  COUNT ( ORANGES :INTEGER) return INTEGER ;
function  COUNT ( APPLES : BIT ) return INTEGER;
```

Cả 2 function là overloaded khi chúng có tên trùng nhau là COUNT . Khi gọi 1 function thực thi , nó có thể nhận dạng chính xác function cần thực thi nhờ kiểu của các số thực đưa vào , bởi vì chúng có kiểu các tham số khác nhau . Ví dụ gọi:

COUNT (20)

Nó được chuyển đến function thứ nhất , tại vì 20 là kiểu INTEGER , trong khi function call :

COUNT (‘1’)

Nó được chuyển đến function thứ hai vì kiểu số thực là BIT .

Nếu 2 chương trình con overloaded có các kiểu tham số và các kiểu kết quả giống nhau , thì chương trình con có thể bị che bởi chương trình con khác . Điều này có thể xảy ra ngẫu nhiên , nếu chương trình con được khai báo bên trong phạm vi của chương trình con khác . Sau đây là ví dụ :

```
architecture  HIDING of DUMMY_ENTITY is
     function ADD (A,B:BIT_VECTOR) return BIT_VECTOR is
     begin
```

```
        --body of function here .
    end ADD;
begin
    SUM_IN_ARCH <= ADD(N1,N2);
    process
        function ADD(C,D:BIT_VECTOR) return BIT_VECTOR is
        begin
            --body of function here.
        end ADD;
    begin
        SUM_IN_PROCESS <= ADD(IN1,IN2);
        SUM_IN_ARCH <= HIDING_ADD .ADD(IN1,IN2);
    end process;
end HIDING;
```

Hàm ADD được khai báo bên trong thân architecture bị che bởi process , vì function ADD thứ hai được khai báo bên trong phần khai báo của process . Function này có thể nằm im ở đường vào bởi bị hạn chế tên function với tên architecture , ta hãy xem phát biểu thứ hai trong process .

Có thể cho 2 chương trình con overloaded đưa vào trực tiếp bên trong 1 vùng , sử dụng bằng mệnh đề use . Trong trường hợp này , nếu nó không thể định rõ chương trình con overloaded nào bị gọi, chương trình con gọi có thể không rõ ràng , và vì thế bị lỗi. Xem ví dụ sau :

```
package P1 is
    function ADD (A,B:BIT_VECTOR) return BIT_VECTOR;
end P1;
package P2 is
    function ADD (X,Y:BIT_VECTOR) return BIT_VECTOR;
end P2;

use WORK.P1.all, WORK.P2.all;
architecture OVERLOADED of DUMMY_ENTITY is
begin
    SUM_CORRECT <= ADD (X=> IN1 , Y => IN2);
    SUM_ERROR   <= ADD (IN1,IN2);
end OVERLOADED;
```

Function gọi trong phát biểu thứ nhất là không bị lỗi , từ đó nó chỉ đến function khai báo trong gói P2 (các tham số hình thức được xác định rõ ràng trong mối liên kết) . trong khi đó phát biểu gán tín hiệu thứ hai không rõ ràng và vì thế bị lỗi .

Hai chương trình con overloaded có thể có số các tham số khác nhau , nhưng có các kiểu tham số và các kiểu kết quả giống nhau . Trong trường hợp này , số các số thực đáp ứng trong chương trình con gọi sẽ nhận dạng chương trình con tương ứng .

Sau đây là ví dụ của tập hợp các số nguyên 2,4,8 :

```
Function SMALLEST (A1,A2 :INTEGER) return INTEGER ;
Function SMALLEST (A1,A2,A3,A4 :INTEGER) return INTEGER ;
Function SMALLEST (A1,A2,A3,A4,A5,A6,A7,A8 :INTEGER)
        return INTEGER ;
```

Function gọi là

```
… SMALLEST (4,5)…
```

Sẽ chỉ đến function thứ nhất , trong khi function gọi :

… SMALLEST (20,45,52,1,89,67,91,22)…

Sẽ chỉ đến function thứ ba . Sự linh động này giúp cho việc viết code được dễ dàng giải mã , từ đó tên các chương trình con giống nhau có thể được đáp ứng khác nhau khi sử dụng tập hợp các input khác nhau .

Gọi đến 1 chương trình con overloaded là không rõ ràng và xuất hiện lỗi , nếu nó không thể nhận dạng chính xác chương trình con bị gọi , sử dụng các thông tin sau :

1. Tên chương trình con (Subprogram name) .
2. Số các số thực (Number of actuals).
3. Kiểu và thứ tự của các số thực (Type and order of actuals).
4. Các tên và tham số hình thức (nếu sử dụng sự liên kết tên).
5. Kiểu kết quả (result type).

Chú ý rằng overloaded không thể phân biệt được các kiểu con . Sau đây là ví dụ :

```
type  LOGIC5 is (‘X’,’0’,’1’,’D’,’Z’);
subtype LOGIC5_01 is LOGIC5 range ‘0’ to ‘1’ ;

function RISING_EDGE ( signal CLOCK : LOGIC5) return BOOLEAN;
function RISING_EDGE ( signal CLOCK : LOGIC5_01) return BOOLEAN;
```

Khai báo function thứ hai bị che bởi function thứ nhất , vì cả hai có các kiểu cơ bản của tham số giống nhau và các kiểu trả về cũng giống nhau . như vậy khi gọi :

```
signal CK : LOGIC5_01;
 …
… RISING_EDGE (CK)…
```

sẽ bị lỗi .

8.3 ĐIỀU KHIỂN OVERLOADING :

Toán tử overloading là một trong những nét đặc biệt của ngôn ngữ . Khi 1 ký hiệu toán tử chuẩn có tác dụng sai lên kiểu của các toán hạng của nó , toán tử được diễn đạt là overloaded . Điều rất cần thiết cho toán hạng overloading xuất hiện từ sự kiện được toán tử định nghĩa trước, trong ngôn ngữ , định nghĩa các kiểu cho các toán hạng . Ví dụ tác dụng của and định nghĩa cho các argument của kiểu BIT và BOOLEAN , và kích thước của chúng là dãy 1 phần tử . Nếu argument thuộc kiểu MVL (MVL là kiểu liệt kê với các giá trị ‘U’,’0’,’1’,’Z’). Trong trường hợp này , nó có thể làm tăng tác dụng của and như 1 function tác dụng lên các argument của kiểu MVL . Toán tử and khi đó được diễn đạt cho overloaded . Toán tử trong biểu thức :

S1 and S2

S1 và S2 là kiểu MVL , chỉ đến tác dụng của and được định nghĩa bằng kiểu viết như 1 function . Toán tử trong biểu thức :

CLK1 and CLK2

CLK1 và CLK2 là kiểu BIT , chỉ đến toán tử and định nghĩa trước .

Các thân function được viết lại để định nghĩa hành vi của các toán tử overloaded . Số lương các tham số trong 1 function phải bằng với số lương dùng với toán tử . Function có nhiều nhất là 2 tham số , tham số thứ nhất bên trái là toán hạng của toán tử , và tham số thứ hai , nếu tồn tại thì nó là toán hạng thứ hai .Sau đây là các ví dụ của khai báo function cho thân function .

```
type MVL is (‘U’,’0’,’1’,’Z’);
function “and” (L,R:MVL) return MVL;
```

```
function "or" (L,R:MVL) return MVL;
function "not" (R:MVL) return MVL;
```

Các toán tử and , or và not là các ký hiệu toán tử được định nghĩa trước , tên function của toán tử overloaded được đặt trong hai dấu ngoặc kép . Để khai báo function overloaded , các toán tử có thể gọi sử dụng 1 trong 2 hai kiểu khác nhau của các ký hiệu :

1. Ký hiệu toán tử chuẩn .
2. Ký hiệu function gọi chuẩn .

Đây là 1 số ví dụ của 2 kiểu ký hiệu cơ bản xuất hiện trên các khai báo function toán tử overloaded :

```
signal A,B,C : MVL;
signal X,Y,Z : BIT ;

    A <= 'Z' OR '1' ;                  -- #1 : standard operator notation.
    B <= 'or' ( '0','Z');              -- #2 : function call notation.
    X <= not Y ;                       -- #3 :
    Z <= X and Y;                      -- #4 :
    C <= ( A or B ) and ( not C );     -- #5 :
    Z <= ( X and Y ) or A;             -- #6 :
```

Toán tử or trong phát biểu đầu tiên chỉ đến toán tử overloaded bởi vì kiểu của toán tử bên trái là MVL . Đây là ký hiệu toán tử chuẩn , do đó ký hiệu toán tử overloaded xuất hiện như ký hiệu toán tử chuẩn . Một ví dụ của ký hiệu function gọi xem trong phát biểu thứ hai , với function overloaded or được gọi một cách rõ ràng . Các toán tử trong phát biểu thứ ba và tư chỉ đến các toán tử khai báo trước vì các toán hạng là kiểu BIT . Phát biểu thứ sáu sẽ bị lỗi , toán tử or không phải là overloaded , nó được định nghĩa với tham số thứ nhất kiểu BIT và tham số thứ hai là kiểu MVL.

Ví dụ cuối cùng là điểm cần quan tâm . Trong các function toán tử overloaded , nó không cần hai toán hạng có cùng kiểu . Trong trường hợp trước , nếu function overloaded or có khai báo khác là :

```
Function "or" ( L : BIT ; R : MVL ) return BIT ;
```

Thì phát biểu thứ sáu sẽ không bị lỗi .

Trường hợp này chủ yếu thường sử dụng cho các vector, do nó cung cấp khả năng thực thi tính toán trên các vector . Sự tính toán trên các vector không được định nghĩa trước trong ngôn ngữ . Các function overloaded cần phải định nghĩa lại . Sau đây là 1 số ví dụ của khai báo toán tử overloaded :

```
function "+" (OPD1,OPD2 : BIT_VECTOR ) return BIT_VECTOR;
function "-" (OPD1,OPD2 : BIT_VECTOR ) return BIT_VECTOR;

type MVL is ('U','0','1','Z');
type MVL_VECTOR is array ( NATUAL range <>) of MVL;
function "+" (OPD1,OPD2 : MVL_VECTOR ) return MVL_VECTOR;
function "-" (OPD1,OPD2 : MVL_VECTOR ) return MVL_VECTOR;
```

Các ví dụ gọi đến các function overloaded như sau :

```
variable   A,B,C,CAB : BIT_VECTOR ( 3 downto 0 );
variable D,E,F,FED : MVL_VECTOR ( 0 to 7);
…
```

```
CAB := A+B-C ;
FED := D – F + E ;
```

Trong phát biểu gán biến đầu tiên , các argument cho các toán tử + và – là vector BIT ; do đó chúng sẽ gọi thực thi các function overloaded “+” và “-“ tương ứng , cho việc định lương tương ứng . Cũng như thế trong phép gán thứ hai , toán tử + và – tác dụng trên vector MVL ; do đó để định lượng , chúng sẽ gọi các function toán tử overloaded “+” và “- “ tương ứng .

8.4 CÁC KÝ HIỆU (signature):

Trong các version trước của ngôn ngữ , nó không thể nhận dạng duy nhất 1 chương trình con overloaded hoặc bảng chử liệt kê overloaded . Ví dụ :

```
--two enumeration type declaration :
type MYBIT is (‘0’,’1’);
type FOUR_VALUE is (‘U’,’0’,’1’,’Z’);

--two overloaded operator function declaration:
function “+” (A,B : BIT_VECTOR) return BIT_VECTOR ;
function “-” (A,B : MVL_VECTOR) return MVL_VECTOR ;
```

Trong các khai báo kiểu liệt kê . ‘0’ và ‘1’ các chử liệt kê overloaded , trong khi trong 2 khai báo function ,“+” là 1 function toán tử overloaded . Nếu cần định nghĩa 1 alias cho function “+” đầu tiên hoặc 1 attribute cho tín hiệu (Các alias và các attribute sẽ được bàn luận trong chương 10), nó không thể nhận ra duy nhất 1 function “+” trong version trước của ngôn ngữ . Tương tự nó không thể nhận dạng ra chử liệt kê ‘0’ và ‘1’ , 1 attribute có thể liên kết với 1 alias được tạo cho 1 trong 2 chử đó .

Các *signature* cung cấp phương pháp phân biệt các chương trình con overloaded và các chử liệt kê overloaded . Một *signature* xem như là các kiểu tham số và kiểu kết quả của 1 chương trình con overloaded hoặc 1 chử liệt kê overloaded . Một signature có dạng :

[*first_parameter_type , second_parameter_type, …*
[return *function_return_type*]]

Chú ý phía ngoài dấu ngoặc vuông ([]) là phần cú pháp và các item không được mô tả . Đây là các signature khai báo function overloaded “+” :

```
[ BIT_VECTOR, BIT_VECTOR return BIT_VECTOR ]
[ MVL_VECTOR, MVL_VECTOR return MVL_VECTOR ]
```

Để xác định tham số và các kiểu kết quả của 1 chử liệt kê , chử liệt kê giải quyết giống như 1 function không có tham số , với tên function giống như chử liệt kê và kiểu trả về của function là kiểu liệt kê . Ví dụ một function chử liệt kê ‘0’ được khai báo phần trước trong kiểu MYBIT và FOUR_VALUE là :

```
function ‘0’ return MYBIT;
function ‘0’ return FOUR_VALUE;
```

Do đó các signature cho mỗi một chử liệt kê là :

```
[ return MYBIT ]
[ return FOUR_VALUE ]
```

Các signature có thể sử dụng trong khai báo alias , đặc biệt là attribute và tên attribute nhận ra duy nhất 1 chương trình con overloaded hoặc 1 chử liệt kê overloaded . Các ví dụ có thể xem trong chương 10.

Chương 9: Thư viện và các gói.

Chương này giải thích package và khả năng biên soạn thiết kế và có khả năng thiết kế thư viện. Nó giải thích ý nghĩa của những thông tin có sẵn trong thư viện và tham gia vào một số desing unit.
Một package cung cấp một sự thuận lợi cho vận hành tới thông tin lưu trữ và có thể dùng chung nhiều desing unit. Một package có thể đặc trưng bởi.

1. Công bố 1 gói và khả năng tùy chọn.
2. Một gói chính

9.1 Packages Declaration:

Một gói đã được xác định (công bố) bao gồm một tập hợp của các declaration chúng có thể tham gia vào nhiều desing unit. Khả năng rõ ràng của package đó là độ rõ ràng của những mẫu tin, chúng có thể thấy sự thành công từ desing unit, để thấy được nó ví dụ như việc xác định được hàm. Một gói chính trong sự bắt được, bao gồm những chi tiết ẩn của một gói được thấy. Ví dụ như một hàm chính.
Cú pháp của gói được xác định là:

```
package package-name is
    package-item-declartions --> These may be:
            -- subprogram declartions
            -- type declartions
        -- subtype declartions
            -- constant declartions
        -- singal declartions
                -- variable declartions
        -- file declartions
                -- alias declartions
        -- component declartions
        -- attribute declartions
```

```
        -- attribute specifications
                -- disconnection specifications
        -- use clauses
    end [package] [package-name];
```
Ví dụ về xác định của một gói phát biểu sau:
```
package SYNTH_PACK is
    constant LOW2HIGH: TIME:= 20 ns;
        type ALU_OP is (ADD, SUB, MUL, DIV, EQL);
                attribute PIPELINE: BOOLEAN;
                type MVL is ('U', 'O', '1', 'Z');
                type MVL_VECTOR is array (NATURAL range <>) of MVL;
                subtype MY_ALU_OP is ALU_OP range ADD to DIV;
                component NAND2
                        port (A, B: in MVL; C: out MVL);
                end component;
        end SYNTH_PACK;
```
Những mẫu tin được xác định trong một gói được xác định có thể là sự nhận được bởi những phần thiết kế khác (units desing) bởi sử dụng thư viện (library) và sử dụng mệnh đề, thông thường xác định tập hợp này có thể kể cả hàm và (function) thủ tục (procedure) và hằng số. Trong trường hợp này việc chạy các chương trình con và các giá trị của các hằng số được xác định tách biệt trong desing unit còn được gọi là package body. Từ đó cung cấp ví dụ về gói không bao gồm các chương trình con hoặc các định nghĩa xác định bằng, một thành phần chính của gói không đòi hỏi phải như vậy.
Xem xét việc xác định gói sau đây.
```
        use WORK.SYNTH_PACK.all
package PROGRAM_PACK is
                constant PROP_DELAY: TIME;              -- A deferred constant.
                function "and" (L, R: MVL) return MVL;
                procedure LOAD (singal ARRAY_NAME: inout MVL_VECTOR;
                                START_BIT, STOP_BIT, INT_VALUE: in INTEGER);
        end package PROGRAM_PACK;       -- The keyword package after end
                                                                        --  is
optional.
```
Trong trường hợp này thành phần chính của gói là điều kiện cần thiết bởi vì xác định gói bao gồm khả năng chính nhận được một hàng và hai chương trình con.

9.2 Package Body (thành phần chính gói):

Một thành phần chính của gói chủ yếu gồm việc chạy chương trình con và giá trị chấp nhận của hằng trong một thành phần của gói. Do vậy nó có thể bao gồm các thành phần khác nữa. Cú pháp của phần chính gói được trình bày như sau:
```
                package body package-name is
                        package-body-item-declartions --> These are:
                                -- subprogram bodies
                                -- complete constant declartions
                                -- subprogram declartions
                                -- type and subtype declartions
                                -- file and alias declartions
                                -- use clauses
                end [package body] [package-name];
```
Tên của gói được phép giống tên của package declartion. Một phần chính gói không cần thiết nếu nó liên kết (kết hợp) package declartion khi không có các chương trình con và hằng. Sự kết hợp phần chính gói để thay thế cho package declartion. PROGRAM_PACK được miêu tả cung cấp trong đoạn sau là:
```
                package body PROGRAM_PACK is
```

```
use WORK.TABLES.all;
constant PROP_DELAY: TIME:= 15ns;
function "and" (L, R: MVL) return MVL is
bigin
        return TABLE_AND (L, R);
        --TABLE_AND is a 2-Dc defined in another package,
        --TABLE.
end "and";
procedure LOAD (singal ARRAY_NAME: inout  MVL_VECTOR;
        START_BIT, STOP_BIT, INT_VALUE: in INTEGER) is
        --Local declartions here.
                Bigin
                        --Procedure behavior here.
                end LOAD;
        end PROGRAM_PACK;
```

Một mẫu tin được có bên trong một phần chính của gói, nó có những hạn chế với thành phần chính và cũng không thể nhận thấy được trong những desing unit khác. Sự hạn chế nó ngăn cản các mẫu tin trong các gói, khi máy có thể nhận được bởi desing unit khác. Do vậy một thành chính của gói là sử dụng tới thông tin mật có sẵn, chúng sẽ không thể nhận thấy được, khi 1 gói sử dụng thông tin về thông tin toán cục, với sự thiết kế khác có thể nhận được. Đây là sự mô phỏng đến một thực thể với một phần chính của kiến trúc khi mà không thể nhận thấy nó được hoạt động, khi mà mẫu tin được xác định trong thực thể được xét có thể nhận thấy nó được trong các phần thiết kế khác. Một cái sự khác biệt giữa xác định một gói và xác định thực thể. Đó là một thực thể có thể tổng hợp của các kiến trúc với các tên khác nhau, trong khi đó một gói được xác định thì có thể ở tại một phần chính của gói, bởi vì tên của chúng có sự ràng buộc.

9.3 Desing File (thiết kế tập tin):

Thiết kế file nguồn của VHDL là các tập tin của ASCII, nó có thể bao gồm một hoặc nhiều đoạn thiết kế, khi một đoạn thiết kế sẽ có các phần sau:

- Entity declartion (xác định thực thể)
- Architecture body (phần chính của kiến trúc)
- Configuration declartion (xác định mô hình sắp xếp)
- Package declartion (xác định gói)
- Package body (phần chính của gói)

Trong phần 9.1 trình bày sự biên dịch sử lý. Tập tin thiết kế dựa vào phân tích VHDL, sau khi cú pháp và phát biểu của tập tin nguồn, nó sẽ đại diện cho việc biên dịch cho mỗi tập tin thiết kế trong tập tin đưa vào. Mỗi Inter khung thông tin nhập vào kho thông tin trong một thư viện tạo mẫu, chúng có được thiết kế và làm việc tại thư viện.

9.4 Desing Libraries (thiết kế thư viện):

Biên soạn thiết kế thông tin có sẵn ở trong thiết kế, thiết kế thư viện. Một thư viện có sẵn là có một vùng thông tin có sẵn của hộ thông của . Theo hướng chỉ tạo của ng thông tin này không bắt buộc phải định nghĩa bởi ngôn ngữ. Trong một sự sắp xếp của hệ thống khi mà thiết kế thư viện ở hệ thống tập tin thư mục và có thể tham khảo những đoạn thiết kế là những thông tin có sẵn của các tập tin trong thư mục này, phần quản lý chính của thiết kế thư viện không được rõ ràng bởi ngôn ngữ và lập lại để xác định việc thi hành lại các lệnh. Số của thư viện thiết kế có toàn quyền quyết định, mỗi thư viện thiết kế có 1 tên logic nó tượng trưng cho một VHDL. Sự kết hợp tên logic với tên vật lý được duy trì trong môi trường chính. Có một thư viện thiết kế với tên logic là STD. Thư viện này bao gồm biên soạn miêu tả trong hai gói STANR và TEXTIO. Một thư viện kế tiếp đó là thư viện có tên logic là WORK. Khi 1 tập tin thiết kế được dịch thì nó sẽ tự đưa vào trong thư viện WORK. Bởi vậy trước khi biên dịch, tên logic của WORK chỉ đến một thư viện thiết kế. (xem hình 9.1)

Thư viện khác nữa gọi là IEEE. Thư viện này bao gồm gói STD_LOGIC_1164 với độ rõ ràng về kiểu logic và sự kết hợp với các hàm khác. Một gói tên chuẩ nữa của IEEE là IEEE std 1164-1993. VHDL nguồn có

mặt trong một tập tin thiết kế. Một tập tin thiết kế có thể bao gồm một hoặc nhiều loại thiết kế. Bài thiết kế là sự thêm vào của các lớp được trình bày như sau:
1. **Primary unit** (một tiêu chuẩn chính): Những cái này không có khả năng để trực tiếp đến các đoạn thiết kế khảo chứng là:
a. **Entity declartion** (khai báo thực thể): Khai báo các mục trong một thực thể là sự ngầm định và có thể nối kết bên trong kiến trúc chính.
b. **Package declartion** (khai báo gói): khai báo các mục với một khai báo gói có thể đưa ra các bài thiết kế khác để sử dụng thư viện và sử dụng các mệnh đề. Khai báo các mục bên trong một khai báo gói cũng là ngầm định tương ứng với phần chính gói.
c. Configuration declartions (khai báo định ng)
2**. Seconry units** (những phần thứ yếu): những phần này không thể bỏ qua được của những phần chính. Những phần thiết kế không bỏ qua được việc khai báo các mục phía trong chúng để đưa ra ngoài phần thiết kế, những phần này không thể tham khảo trong những phần thiết kế khác.
Những phần thứ yếu là:
a**. Architecture bodies:** Khai báo một tín hiệu trong phần chính của kiến trúc không thể tham khảo trong những phần thiết kế khác.
b**. Package bodies (các phần chính gói):** Một phần thứ yếu có thể đúng (chính xác) cho một tên trong thư viện. Các phần thứ yếu có thể kết hợp với những phần chủ yếu khác có những tên đúng như tên trong thư viện. Như vậy, phần thứ yếu có thể được phép giống về tên để kết hợp phần chủ yếu. Như trong ví dụ được xét, cho rằng gọi một thực thể AND_GATE có thể trùng tên trong thư viện. Với phần chính của kiến trúc thì nó có phép giống như tên, và khác thực thể MY_GATE vì vậy trong thư viện thiết kế phần chính của kiến trúc được phép có tên AND_GATE.
Những phần thứ yếu được phép cùng tồn tại với những phần chủ yếu trong thư viện, ví dụ chư khai báo thực thể và tất cả phần chính kiến trúc của nó được thường trú trong thư viện.
Tương tự, khai báo một gói và phần chính gói được phép kết hợp thường trú trong một thư viện độc lập.
Kai báo định ng là một phần chủ yếu, nó thường trú trong thư viện là định ng khai báo thực thể.

9.5 Order of Analysis (phân tích sự đúng đắn):

Từ đó nó có thể đưa đến khai báo các mục trong những phần chính tới những thiết kế khác, phải bắt buộc sự kế tiếp nhau trong mục thiết kế, một thiết kế tham khảo đến việc khai báo các mục trong phần chính khác, có thể được phân tích sau phần chủ yếu. Ở ví dụ, nếu việc khai báo định ng tham khảo một thực thể COUNTER, khai báo thực thể với COUNTER được phân tích sau khai báo định ng.
Một phần chủ yếu được phân tích sau nhiều cái sự kết hợp những phần thứ yếu. Ví dụ khai báo thực thể được phân tích sau các phần chính kiến trúc.

9.6 Implicit Visibility (trạng thái ngầm định):

Một phần chính của kiến trúc tiếp nhận khảnăng ngầm định của tất cả khai báo trong thực thể, từ đó nó giới hạn các thực thể đó bởi các câu lệnh.

architecture *architecture-name* **of** *entity-name* **is** …

Tương tự, phần chính gói tiếp nhận sự ngầm định ở trong khai báo gói bởi vì phát triển đầu tiên của nó là:

package body *package-name* **is** …

Khi tên gói giống như trong khai báo gói.

9.7 Explicit Visibility (trạng thái chi tiết):

Trạng thái chi tiết hoá, khai báo các mục trong các phần thiết kế khác, phần mềm được sử dụng trong mệnh đề sau.
1. Library clause
2. Use clause
Sử dụng các mệnh đề có thể xuất hiện trong nhiều khai báo bộ phận của một thiết kế. Nếu là một thư viện mệnh đề hoặc sử dụng mệnh đề tại điểm bắt đầu của phần thiết kế, nó gọi mệnh đề ngữ cảnh. Hình 9.2 trình bày một ví dụ. Xác định các mục trong mệnh đề ngữ cảnh thường nhận thấy một bài thiết kế, chúng có thể hỗ trợ tới mệnh đề ngữ cảnh, các phần này không nhận thấy được những đoạn thiết kế chủ yếu (chính), chúng có

thể biến ng giống như một tập tin thiết kế (desing file). Điều muốn nói ở đây là nếu một tập tin thiết kế bao gồm 3 phần thiết kế chính, được trình bày trong hình 9.3, mệnh đề ngữ cảnh được xác định bởi mỗi phần thiết kế nếu cần. Ví dụ mệnh đề ngữ cảnh được xác định trước phần thiết kế A có thể nhận thấy được phần thiết kế A, mệnh đề ngữ cảnh được xác định sau phần thiết kế B có thể nhận thấy được phần B.

9.7.1 Library Clause:

Thư viện các mệnh đề tạo ra sự nhận biết các tên logic của thư viện, có thể tham khảo phía trong một phần thiết kế. ng của một mệnh đề trong thư viện là:

library *list-of-logical-library-names* ;

Mệnh đề của thư viện

library TTL, CMOS;

Tạo ra tên logic TTL và CMOS có thể nhận thấy được trong một phần thiết kế. Chú ý các phần của mệnh đề thư viện không tạo ra những đoạn thiết kế hoặc các mục đó có mặt trong thư viện, nó thường thiết lập (tạo) tên thư viện để nhận biết (nó giống như một khai báo thay thế cho tên thư viện). Ví dụ: muốn sử dụng tên 'TTL.SYNTH_PACK.MVL' phía trong một phần thiết kế khai báo đầu không thể hiện tên thư viện sử dụng mệnh đề "library TTL"

Mệnh đề thư viện

library STD, WORK;

Khai báo thường là được ngầm định trong phần thiết kế.

9.7.2 USE Clause (sử dụng mệnh đề):

Có 2 hình thức chính sử dụng mệnh đề:

use *library-name.primary unit-name* ; --hình thức 1
use *library-name.primary unit-name.* item; --hình thức 2

Hình thức đầu của sử dụng mệnh đề xác định tên của phần chủ yếu (chính) từ thư viện tới việc tham khảo trong mô tả một thiết kế. Xét ví dụ:

```
library CMOS;
use CMOS.NOR2
configuration . . . is
        . . . use entity NOR2 (. . .);
end;
```

Chú ý: thực thể NOR2 được phép có sẵn trong thư viện trước khi thử biên dịch một phần thiết kế sử dụng. Hình thức 2 của sử dụng mệnh đề có thể nhận thấy được khai báo các thành phần chính. Do vậy thành phần đó có thể, tham khảo phía bên trong của phần thiết kế. Ví dụ:

```
library ATTLIB
use ATTLIB.SYNTH_PACK.MVL;
--MVL is a type declared in SYNTH_PACK package.
--The package SYNTH_PACK is stored in the ATTLIB desing library.
entity NAND2 is
    port (A, B: in MVL; . . .) . . .
```

Nếu tất cả các thành phần ở phía trong một phần chính thì có thể dùng khóa all để sử dụng.Ví dụ:

use ATTLIB.SYNTH_PACK.**all**;

Thiết lập khao báo tất cả các thành phần ở gói SYNTH_PACK trong thư viện ATTLIB. Sử dụng mệnh đề

use ATTLIB.**all**

Thiết lập tất cả các tên của một phần chính có mặt trong thư viện ATTLIB

Trạng thái bên ngoài của các thành phần tới một đoạn thiết kế có thể nhận được bởi một mong muốn tốt hơn. Một phương pháp là sử dụng chọn tên.

Một ví dụ sử dụng chọn tên là:

```
library ATTLIB;
use ATTLIB.SYNTH_PACK;
entity NOR2 is
    port (A,B: in SYNTH_PACK.MVL; . . .) . . .
```

Vậy chỉ có tên của phần chính là thiết lập sự nhận biết được sử dụng các mệnh đề, tên của phần chính được phép sử dụng liên tiếp (dọc theo) tồn tại tên kiểu tham khảo, đó là SYNTH_PACK.MVL được xác định.Ví dụ khác sẽ trình bày phần tiếp theo.
Kiểu Value_9 là được định nghĩa trong gói SIMPACK khi mà biên dịch tong thư viện CMOS.

```
library CMOS
package P1 is
        procedure LOAD (A, B: CMOS.SIMPACK.VALUE_9; . . .);
end P1;
```

Trong trường hợp này, tên phần chính SIMPACK chỉ được xác định duy nhất một thời gian sử dụng.
Do vậy, chúng TABLE có thể đưa ra các thành phần áp dụng các thư viện. Nếu cần thiết đưa ra các loại thành phần thiết kế chúng giống như trong thư viện ?
Trong trường hợp chúng không cần tới việc xác định một mệnh đề thư viện khi đó thường là một đoạn thiết kế mà nó được khai báo ngầm định trong mệnh đề thư viện.

```
library STD,WORK;
```

Tương tự thư viện STD bao gồm các gói STANRD và TEXTIO. Gói STANRD gồm có các khai báo trên cho các kiểu CHARACTER, BOOLEAN, BIT_VECTOR và INTEGER. Sử dụng các mệnh đề khai báo ngầm định.

```
use STD.STANRD.all;
```

Tất cả các thành phần khai báo phía trong gói TANRD là có sẵn để sử dụng trong cấu trúc của VHDL, tương tự như vậy đối với gói TEXTIO. Nếu khai báo các thành phần trong gói này cần tham khảo đến sử dụng mệnh đề hình thức sử dụng là:

```
use STD.TEXTIO.all;
```

PHầN III:

VIẾT CHƯƠNG TRÌNH THIẾT KẾ MẠCH CỘNG 8 BIT SONG SONG BCD CÓ HAI TOÁN HẠNG

I/ GIỚI THIỆU CHUNG :

-Trong máy tính những phép tính và máy tính thường sử dụng mã BCD biểu diễn số thập phân, mỗi số thập phân thường biểu diễn cho nó bởi 1 group 4 bit thứ tự từ 0000 đến 1001.Việc cộng số thập phân đó được biểu diễn trong các trường hợp sau:

*Tổng lớn hơn hoặc bằng 9:

Khi cộng 2 số 5 và 4 có mã là số BCD :

5 0110 mã BCD của 5
+4 0100 mã BCD của 4

9 1001 mã BCD của 9

Ví dụ : ta cộng 33 với 45

45 0100 0101 ←BCD for 45
+33 0011 0011 ←BCD for 33
78 0111 1000 ← BCD for 78

Trong ví dụ này 4 bit của 5 và 3 thì cộng kết quả ra như cộng nhị phân BCD của 1000 là 8,tương tư mã BCD của 7 là 0111,tổng cộng kết quả là 0111 1000,mà kết quả đó là số BCD for code 78

*Tổng lớn hơn 9: Ví dụ : cộng 7 và 6

7 0110 ← BCD for 7
+6 0111 ← BCD for 6
13 1101 sai kết quả của group BCD này

Tổng 1101 không tồn tại trong mã của BCD ,nó thì phải được sữa sai theo qui tắc sau:

0110 ← BCD for 6
+0111 ←BCD for 7
1101 kết quả sai
0110 ←-sữa sai cộng thêm 6
0001 0110 ←BCD for 13
1 3

Như đã biểu diễn trên ,0110 được cộng để sửa sai cho kết quả đúng BCD.Chú ý : kết quả nhớ của vị trí số thập phận thứ 2 .

Ví dụ khác:

47 0100 0111 BCD for 47
+35 0011 0101 BCD for 35
82 0111 1100 sai kết quả
số nhớ khi sữa 1 0110 cộng thêm 6
1000 0010 tổng đúng BCD

8 2

Khi cộng 4 bit của 7 và 5 kết quả tổng sai và khi sữa sai ta cộng thêm 6 (0110) .Khi đó nhớ ra là 1 mà khi đó kết quả nhớ đó được cộng vào số BCD của vị trí số thập phân thứ 2 .

Thí dụ như:

1 số nhớ khi cộng ra kết quả ban đầu đem lên
59 0100 1001 ← BCD for 59

+38 0011 1000 ←BCD for 38
97 1001 0001
0110 cộng sữa sai cộng thêm 6
1001 0111 BCD for 97

9 7

II/ NGUYÊN LÝ CỘNG 2 SỐ BCD 8 bit :

-Khi tổng nhỏ hơn 9 thì ta lấy kết quả là kết quả cộng ban đầu
-Khi tổng lớn hơn 9 ta lấy kết quả ban đầu cộng thêm "0110"(6) để sữa sai
-Và kết quả có nhớ sẽ được đưa vào group đầu cộng ra kết quả của group đầu.
-Sau cùng là kết quả chung ghép của 2 group lại với nhau để ra 1group 8bit bcd
-Có một trường họp khác là sau khi cộng sữa sai xong ta mới có nhớ thì ta đem lên cộng tất cả các hàng đó lại với nhau kết quả ra đưa group đầu tiên

III /LƯU ĐỒ CỦA CHƯƠNG TRÌNH :{mạch cộng 8 bit song song BCD cho 2 toán hạng }

IV /CHƯƠNG TRÌNH :

Chương trình thể hiện qua hai phần mềm 1 cho Leonardo thể hiện dạng mô phỏng mạch ban đầu,1 cho Max+plus2,trong đó Max+plus2 cho kết quả ra dạng sóng đã có kiểm tra kết quả dạng sóng đúng thông qua các mạch như sau:

Mạch đã mô phỏng theo Leonardo:

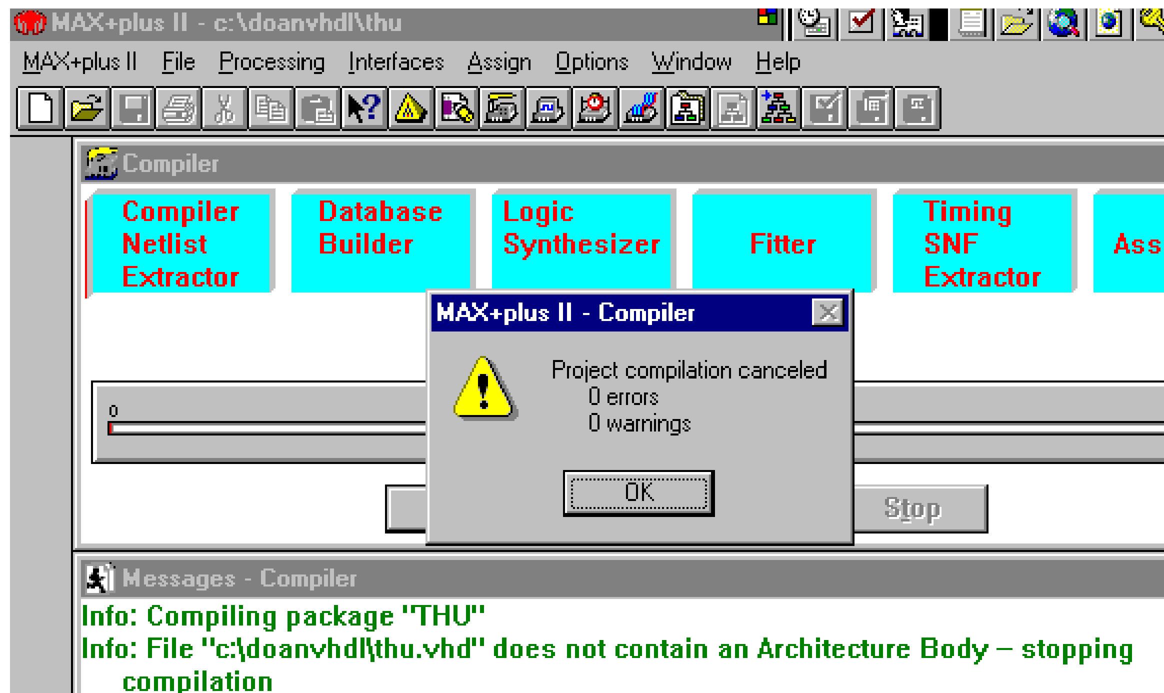
Mạch đã test theo max+plus2

Phần chương trình:Có 2 cách viết

***Cách 1*: theo kiểu behaviour(hành vi)**

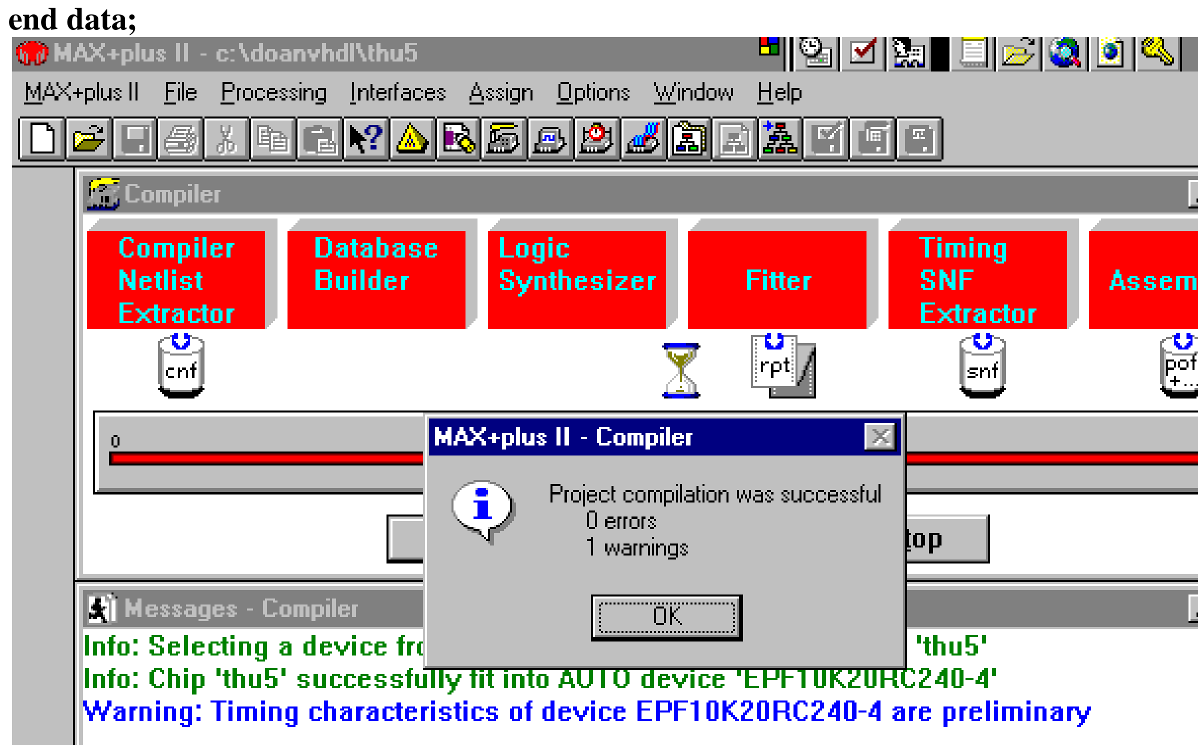
```
package thu is {package này dùng chung cho 2 cách Behaviour và structure}
  type myinteger is range 1 to 3;
  function add (A,B : bit_vector(3 downto 0);cin : bit) return bit_vector;
end thu;
package body thu is
   function add (A,B :bit_vector(3 downto 0);cin :bit) return bit_vector is
          variable R: bit_vector(4 downto 0);
      variable C : bit_vector(3 downto 0);
                begin
                        R(0) := A(0) xor B(0) xor Cin;
                        C(0) := (A(0) and B(0)) or (A(0) and Cin) or (B(0) and Cin);
                        for i in 1 to 3 LOOP
                        R(i) := A(i) xor B(i) xor C(i-1);
                        C(i) := ((A(i) xor B(i)) and C(i-1)) or (B(i) and A(i));
                        end LOOP;
                        R(4) := C(3);
                        return R;
                end ;
end thu;
```

Sau khi sủ dụng công cụ soạn thảo –text- editor file và dịch thành công ta có mạch:

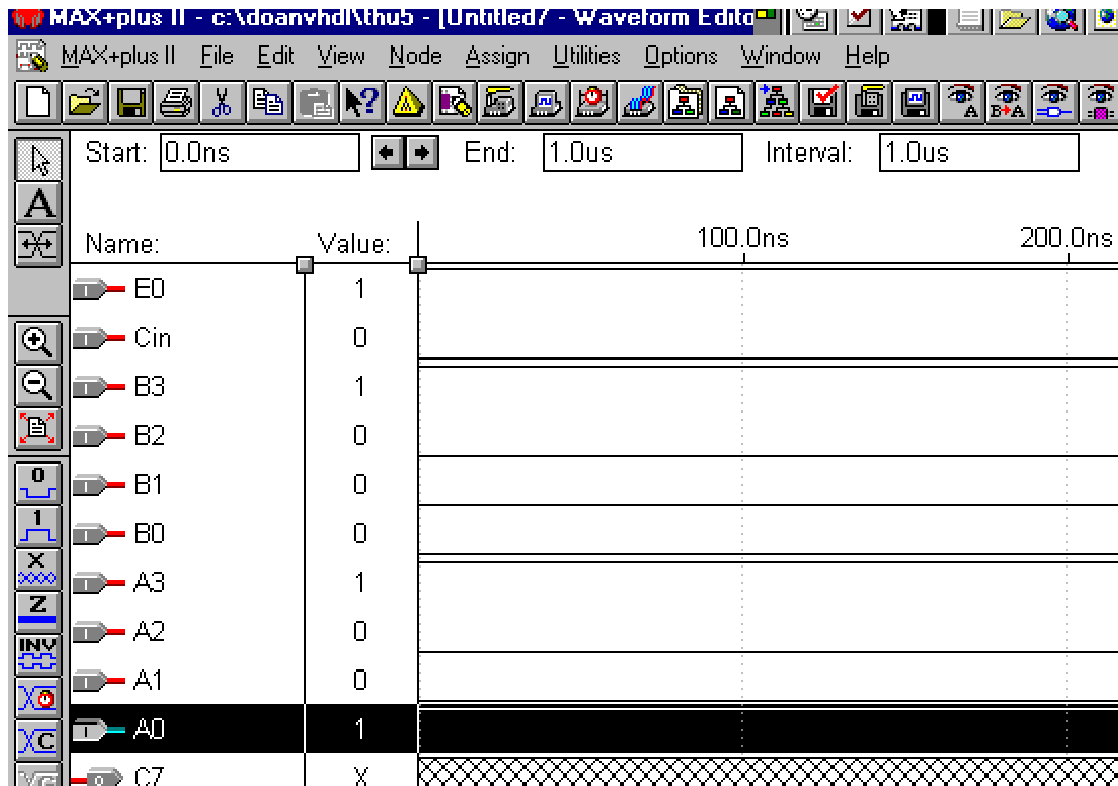



```
library ieee;-- chương trình viết theo kiểu behaviour
use ieee.std_logic_1164.all;
library work;
use work.thu.all;
 entity thu5 is
        port (A,B,E,F: in bit_vector(3 downto 0);
                Cin :bit;
                C : out bit_vector(7 downto 0);
                Cout : out bit);
  end thu5;
architecture data of thu5 is
        signal out1,out2: bit ;
        signal C1,C2: bit_vector(3 downto 0);
        begin
        P: process(A,B,Cin)
                variable tam1,X: bit ;
                variable tam: bit_vector(3 downto 0);
                variable R,R1,R2,R3: bit_vector(4 downto 0) ;
                begin
                tam:= "0110";
                out1 <= '0';
                R:= add(A,B,Cin);
                X:= R(4) or (R(3) and (R(2) or R(1)));
                if X='1' then
                R1:= add(R(3 downto 0),tam,out1);
                C1<=R1(3 downto 0);
                out2<= R1(4) or R(4);
                else
```

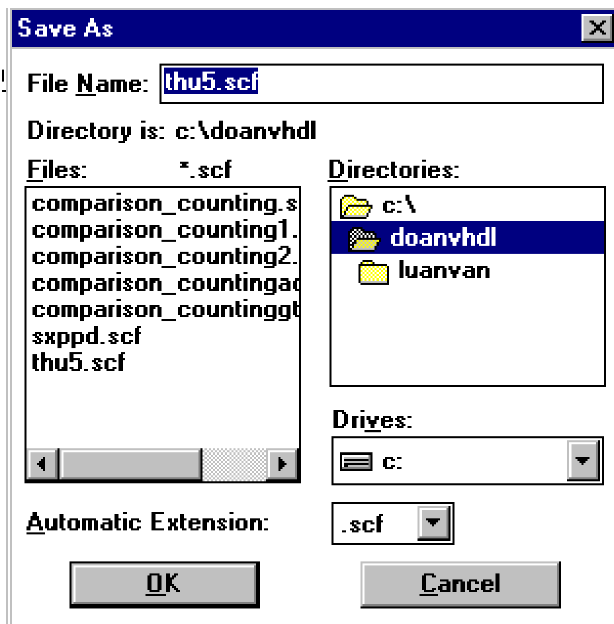
```
            C1 <= R(3 downto 0);
            out2 <= R(4);
            end if;
            R2:=add(E,F,out2);
            X:=R2(4) or (R2(3) and (R2(2) or R2(1)));
            if X='1' then
            R3:= add(R2(3 downto 0),tam,out1);
            C2<= R3(3 downto 0);
            Cout<=R3(4) or R2(4);
            else
            C2<=R2(3 downto 0);
            Cout<=R2(4);
      end if;
      C<= C2&C1;
      end process;
end data;
```

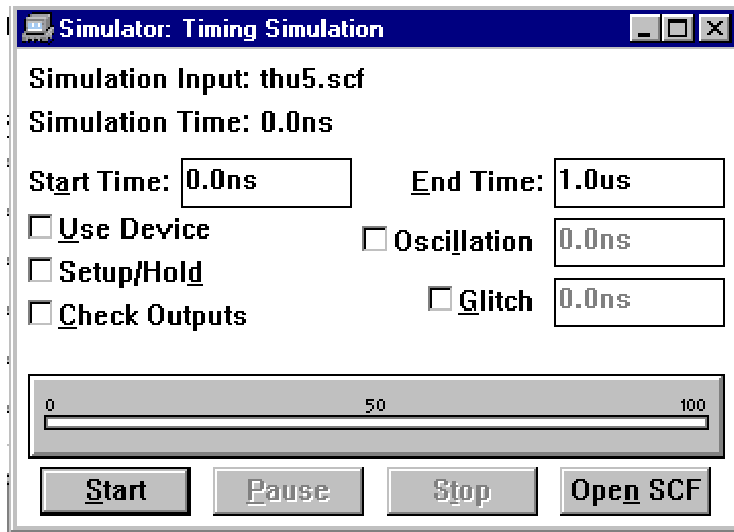


Sau khi dịch thành công ta đưa vào dạng sóng của các biến và nhập vào các thông số cần thiết

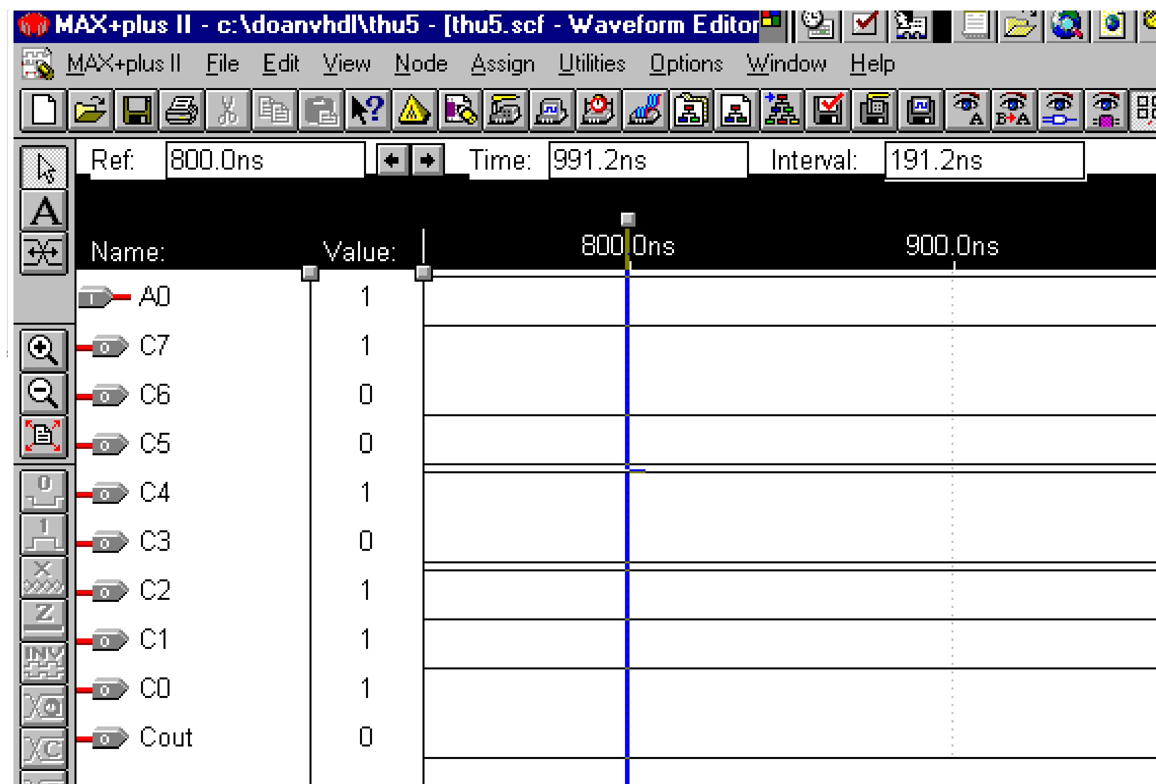


Xong ta bắt đầu lưu lại theo trình tự sau và simulation nó theo các hình vẽ:





Đáp số của các biến hay kết quả kiểm tra sóng ra như hình sau:



Cách 2: **Chương trình viết theo kiểu Structure**(port map):

```
 package thu is
   type myinteger is range 1 to 3;
   function add (A,B : bit_vector(3 downto 0);cin : bit) return bit_vector;
end thu;
package body thu is
    function add (A,B :bit_vector(3 downto 0);cin :bit) return bit_vector is
```

```
            variable R: bit_vector(4 downto 0);
        variable C : bit_vector(3 downto 0);
                begin
                        R(0) := A(0) xor B(0) xor Cin;
                        C(0) := (A(0) and B(0)) or (A(0) and Cin) or (B(0) and Cin);
                        for i in 1 to 3 LOOP
                        R(i) := A(i) xor B(i) xor C(i-1);
                        C(i) := ((A(i) xor B(i)) and C(i-1)) or (B(i) and A(i));
                        end LOOP;
                        R(4) := C(3);
                        return R;
                end ;
end thu;
```

(chương trình cộng bcd 4bit)

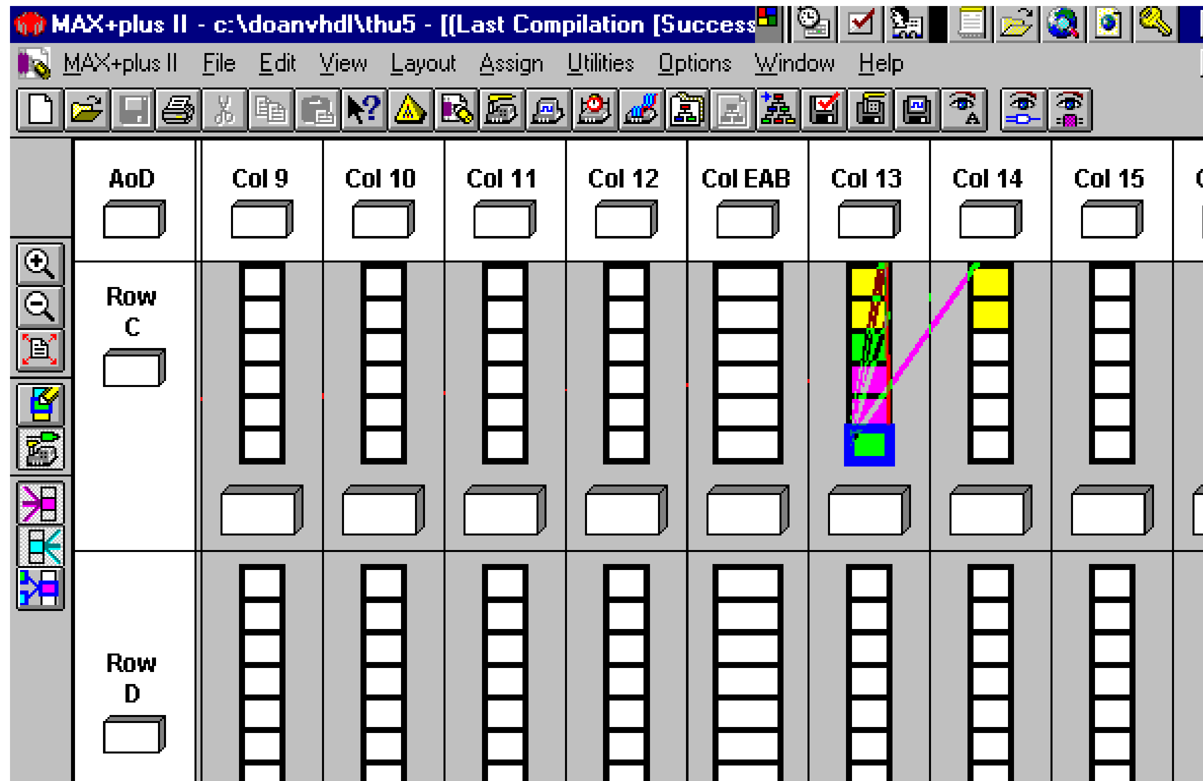
```
library ieee;
use ieee.std_logic_arith.all;
library work;
use work.thu.all;
 entity bcd is
        port (A,B: in bit_vector(3 downto 0);
                Cin :bit;
                C : out bit_vector(3 downto 0);
                Cout : out bit);
  end bcd;
architecture data of bcd is
        signal out1,out2: bit ;
        signal C1,C2: bit_vector(3 downto 0);
        begin
        P: process(A,B,Cin)
                variable tam1,X: bit ;
                variable tam: bit_vector(3 downto 0);
                variable R,R1,R2,R3: bit_vector(4 downto 0) ;
                begin
                tam:= "0110";
                out1 <= '0';
                R:= add(A,B,Cin);
                X:= R(4) or (R(3) and (R(2) or R(1)));
                if X='1' then
                R1:= add(R(3 downto 0),tam,out1);
                C<=R1(3 downto 0);
                Cout<= R1(4) or R(4) ;
                else
                C <= R(3 downto 0);
                Cout <= R(4);

                end if;
        end process;
end data;
```

(chương trìng cộng BCD 2 group 4→ 8bit BCD)

```
entity tuan is
port(A,B,E,F:in bit_vector(3 downto 0);
       Cin:bit;Cout:out bit;
      C:out bit_vector(7 downto 0));
end ;
architecture detai of tuan is
            signal C1,C2: bit_vector(3 downto 0);
            signal tam: bit;
Component bcd
    port (A,B: in bit_vector(3 downto 0);
            Cin:bit;C:out bit_vector(3 downto 0);
            Cout :out bit);
   end component;
   begin
            l1:bcd port map(A,B,Cin,C1,tam);
            l2:bcd port map(E,F,tam,C2,Cout);
        C<=C2&C1;
  end detai;
```

Phần Floor plan editor và thời gian trể ma trận input,output(delay matrix) các ngõ ra cũa các Cout,Cin và kết quả ra C0→C7



MAX+plus II - c:\doanvhdl\thu5 - [Timing Analyzer]

MAX+plus II File Node Analysis Assign Utilities Options Window Help

Delay Matrix

Destination

Source	Cout	C0	C1	C2	C3
A0	35.5ns/52.3ns	17.9ns	25.4ns/35.5ns	25.5ns/34.1ns	25.5ns/31.0ns
A1	35.9ns/49.0ns		22.1ns/32.2ns	22.2ns/30.8ns	22.2ns/28.5ns
A2	35.4ns/46.1ns		25.5ns/29.2ns	21.0ns/29.3ns	21.0ns/29.3ns
A3	32.3ns/41.5ns		22.3ns/24.7ns	22.4ns/23.3ns	20.2ns/22.4ns
B0	34.9ns/51.7ns	17.3ns	24.8ns/34.9ns	24.9ns/33.5ns	24.9ns/30.4ns
B1	34.9ns/48.6ns		21.7ns/31.8ns	21.8ns/30.4ns	21.8ns/28.7ns
B2	32.1ns/43.0ns		22.2ns/26.1ns	17.7ns/26.2ns	17.7ns/26.2ns
B3	29.2ns/38.4ns		19.2ns/21.6ns	19.3ns/20.2ns	17.1ns/19.3ns
Cin	32.4ns/49.2ns	14.8ns	22.3ns/32.4ns	22.4ns/31.0ns	22.4ns/27.9ns
E0	26.2ns/33.1ns				